SOCIEDADE EDUCACIONAL DE SANTA CATARINA - SOCIESC
INSTITUTO SUPERIOR TUPY - IST

MARCELO RENAN BECHER

MONTAGEM DE UM AMBIENTE DE CLUSTER USANDO SOFTWARE LIVRE: UMA ABORDAGEM AOS CLUSTERS DE ALTA DISPONIBILIDADE

Joinville
2007/1

MARCELO RENAN BECHER

MONTAGEM DE UM AMBIENTE DE CLUSTER USANDO SOFTWARE LIVRE: UMA ABORDAGEM AOS CLUSTERS DE ALTA DISPONIBILIDADE

Esse trabalho será apresentado ao Instituto Superior Tupy como requisito parcial para a obtenção de grau de bacharel em Sistemas de Informação, sob orientação do Professor Eduardo da Silva.

PROF. EDUARDO DA SILVA

Joinville

2007/1

MARCELO RENAN BECHER

MONTAGEM DE UM AMBIENTE DE CLUSTER USANDO SOFTWARE LIVRE: UMA ABORDAGEM AOS CLUSTERS DE ALTA DISPONIBILIDADE

Esse trabalho foi julgado e aprovado em sua forma final pela banca examinadora abaixo assinada.

Joinville, 26 de junho de 2007

____
Prof. Eduardo da Silva

____
Prof. MSc. Mehran Misaghi

____
Prof. MSc. Alexandre Lima

BECHER, MARCELO RENAN **MONTAGEM DE UM AMBIENTE DE CLUSTER USANDO SOFTWARE LIVRE: UMA ABORDAGEM AOS CLUSTERS DE ALTA DISPONIBILIDADE**
Joinville: SOCIESC, 2007/1.

A toda minha família por todo o apoio ao longo da caminhada.

AGRADECIMENTOS

A DEUS, que sempre me deu força para continuar nos momentos de fraqueza;
A minha família, pelo apoio e incentivo aos estudos;
Aos amigos Carlos Diego Russo Medeiros e Felipe Nogaroto Gonzalez, por todo o conhecimento que me foi repassado.
A todos que não foram mencionados, mas de alguma forma colaboraram com o desenvolvimento deste trabalho.

O impossível é questão de tempo.

Alberto Satiel

RESUMO

O agrupamento de dois ou mais servidores, de modo que possa ser visto como um único recurso é definido como *cluster*. Essa tecnologia permite que se consiga por exemplo, alto desempenho, balanceamento de carga ou alta disponibilidade em uma aplicação. Este trabalho apresenta os principais tipos de *clusters* existentes hoje no mercado e em que situações podem ser utilizados. É aboradada também a alta disponibilidade, apresentando as classificações e como se obtê-la. Ao final é montado um ambiente de testes, onde são utilizadas as ferramentas de alta disponibilidade presentes no sistema operacional Linux e, também apresentado o estudo da caso de uma transportadora, onde foi criado um ambiente de comunicação seguindo os princípios de alta disponibilidade.

Palavras-chave: Agrupamento. Disponibilidade. Falhas. Cluster

ABSTRACT

The grouping of the two or most servers, in way that can be seen as an only resource is defined as cluster. This technology allows to get for example, high performance, load balancing or high-availability in an application. This paper presents the main types of existing clusters in the market today and where situations each type of cluster can be used. High-availability is also cited, presenting the classifications and how to get high-availability. To the end an environment of tests is mounted, where the tools presents in Linux operational system are used, and presented a case study of a logistics company, where a high-availability communication environment was created following the principles of high-availability.

Keywords: Grouping. Availability. Faults. Cluster

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

As corporações dependem cada vez mais das aplicações e serviços fornecidos pela área de tecnologia da informação para realizar tarefas críticas e para que suas operações, processos e negócios não parem.

Em um ambiente ideal, os recursos computacionais deveriam estar disponíveis 100% do tempo, sem apresentar falhas (WEBER, 2002). Porém as falhas são inevitáveis (PEREIRA, 2004) e vários fatores podem fazer com que um recurso computacional apresente indisponibilidade, tais como catástrofes naturais, falhas em aplicativos ou problemas com *hardware*.

As conseqüências podem ser as mais diversas, desde a perda de dados, comprometimento do negócio da empresa e até mesmo a perda de vidas humanas (PEREIRA, 2004).

O mercado atualmente oferece várias soluções que permitem a criação de uma infraestrutura redundante. Fabricantes de hardware e desenvolvedores de software agregam a seus produtos características que permitem que os recursos computacionais continuem funcionais, mesmo que algum componente envolvido apresente uma falha.

A alta disponibilidade pode ser alcançada de duas formas: por meio do uso de *hardware* proprietário, fabricado exclusivamente para esse fim, ou; por meio do uso de *hardware* comum, em conjunto com aplicativos específicos (TOSATTI, 2006).

Este trabalho apresenta um estudo baseado no uso de *hardware* comum em conjunto com aplicativos conhecidos como *software* livre, os quais podem ser utilizados sem que seja necessário o pagamento por uma licença de uso, como ocorre com aplicativos comerciais. O *hardware* comum e os *softwares* livres implementados em conjunto permitem a criação de um ambiente de alta disponibilidade.

O capítulo 2 apresenta as características que tornam o sistema operacional Linux flexível e possibilitam usá-lo para solucionar problemas nas mais diversas áreas. A seguir são apresentadas algumas definições para o termo *cluster* e apresentados os conceitos abstratos que estão relacionados à eles. Este capítulo também mostra a história de surgimento dos primeiros *clusters*, quais os principais tipos e cita alguns exemplos de áreas em que podem ser utilizados. Ao final é apresentada a arquitetura de *clusters*, segundo a proposta do Padrão de Cluster Aberto - *Open Cluster Framework* (OCF).

O capítulo 3 mostra os conceitos relacionados à alta disponibilidade, e apresenta uma das fórmulas mais utilizadas para se calcular a disponibilidade de um sistema. São mostrados

os principais tipos de disponibilidade que podem ser requeridos em um sistema, e os principais tipos de falhas que podem ocorrer. Ao final do capítulo são apresentadas as fases da tolerância à falhas.

O capítulo 4 apresenta as principais ferramentas, que podem ser utilizadas em conjunto com o sistema operacional Linux na criação de um ambiente de alta disponibilidade, e o capítulo 5 descreve os testes de validação que foram realizados em um laboratório.

Ao fim, o capítulo 6 apresenta o estudo de caso da transportadora Levatudo Logística Ltda, onde foi implementado um ambiente de alta disponibilidade utilizando as ferramentas estudadas neste trabalho.

2 INTRODUÇÃO AOS CLUSTERS

Este capítulo visa a compreensão de uma das áreas da Tecnologia da Informação (TI) que trata do agrupamento de computadores, ou *clusters*. A seção 2.1 apresenta as características do *software* livre. Em seguida o conceito de *cluster* é apresentado de forma mais detalhada, passando pelas abstrações e história de surgimento dos *clusters*.

A seção 2.5 apresenta os principais tipos de *clusters* existentes no mercado e as áreas em que são utilizados. Ao final é apresentada uma proposta de arquitetura de comunicação entre as camadas que formam a arquitetura dos *clusters* e uma conclusão sobre o capítulo 2.

Antes de iniciar a abordagem aos *clusters* é importante entender o que é o *software* livre e como surgiu o sistema operacional Linux.

2.1 O PAPEL DO SOFTWARE LIVRE COMO BASE PARA OS CLUSTERS

Em 1984, o até então pesquisador do Instituto de Tecnologia de Massachusetts - *Massachusetts Institute of Technology* (MIT) Richard Stallman, criou a Fundação do Software Livre - *Free Software Foundation* (FSF), pela qual passou a defender a idéia de que o código fonte de todo e qualquer aplicativo deve estar disponível para que qualquer pessoa tenha acesso. Ele não achava correto que os fabricantes fornecessem todo o aplicativo necessário para o funcionamento do seu equipamento. Stallman sentiu que isso impedia que o aplicativo fosse aperfeiçoado, pois somente algumas poucas pessoas tinham acesso ao código fonte (FERREIRA, 2001).

Foi então que ele começou a defender a idéia de *software* livre. No mesmo ano criou a Licença Pública Geral - *General Public License* (GPL), a qual se caracteriza por permitir quatro liberdades a quem estiver usando o aplicativo (STALLMAN, 2007).

I. utilizar o aplicativo sem nenhuma restrição e sem ter que pagar por uma licença de uso;

II. estudar o aplicativo e entender como ele funciona;

III. redistribuir o aplicativo para outras pessoas;

IV. aperfeiçoar o aplicativo, corrigindo erros ou acrescentando novas funcionalidades.

A partir de então a FSF passou a criar aplicativos e disponibilizá-los sobre a licença GPL, além de divulgar a filosofia mundo afora.

Anos mais tarde, em 1990, o programador finlandês Linus Torvalds inicia como um projeto pessoal, o desenvolvimento do sistema operacional Linux (NEMETH, 2004), o qual foi baseado no sistema operacional Minix, que por sua vez foi criado com base no Unix, desenvolvido na década de 70 e já conhecido por sua estabilidade (PITANGA, 2002).

Torvalds utilizou o compilador Compilador GNU C - *GNU C Compiler* (GCC) criado pela FSF, no desenvolvimento do Linux. Ele também disponibilizou seu projeto na Internet sob a licença GPL, e passou a receber colaboração de código de muitos programadores. Dessa maneira, o sistema tornou-se muito conhecido e começou a ser utilizado no meio acadêmico e nas empresas. Porém, para o sistema ser utilizado em ambientes de missão crítica, ainda faltavam ferramentas que garantissem alta disponibilidade dos serviços (PEREIRA, 2004). No início foi difícil para os fornecedores de soluções baseadas no sistema operacional Linux, pois o Unix originalmente não oferecia recursos que permitissem, de maneira automática, a troca de informações entre computadores, nem o compartilhamento do sistema de arquivos em rede (WEBER, 2002). Aos poucos surgiram projetos com o objetivo de desenvolver ferramentas que pudessem preencher este espaço, deixando o Linux no mesmo nível das soluções comerciais (PEREIRA, 2004).

O Linux cresceu, e hoje é reconhecido pela sua estabilidade, flexibilidade e robustez, aliadas ao baixo custo. É um dos sistemas operacionais mais usados, principalmente em servidores. A filosofia do *software* livre tornou-se conhecida mundialmente, e hoje impulsiona milhares de programadores e empresas de desenvolvimento de aplicativos, nas mais diversas áreas.

2.2 PRINCÍPIOS DE UM CLUSTER

Do inglês, *cluster* significa grupo de coisas do mesmo tipo ou semelhantes, juntar-se, agrupar-se (PARKER, 2007).

O autor Pitanga (2002, p. 12) diz que "quando você utiliza dois ou mais computadores em conjunto para resolver um problema, você tem um *cluster*. "

O autor Pereira (2004, p. 5) descreve da seguinte maneira:

> Um *cluster*, ou aglomerado, é uma coleção de computadores que trabalham juntos para criar um sistema mais poderoso, ou um conjunto de máquinas independentes, que cooperam umas com as outras para atingir um determinado objetivo comum.

Caracteriza-se por *cluster* o ambiente que oferece um recurso computacional por meio de vários computadores interligados por uma rede, que são vistos pelos usuários, aplicativos e outros servidores como um recurso único. Além disso, os usuários não devem saber que estão usando um *cluster* e os demais servidores não devem saber que estão se comunicando com um *cluster* (KOPPER, 2005).

Segundo Kopper (2005, p. 3), as quatro características básicas de um *cluster* são:

I. os usuários não saberem que estão usando um *cluster*

II. os computadores não saberem que eles fazem parte do *cluster*

III. as aplicações em execução, não saberem que fazem parte de um *cluster*

IV. demais computadores que compõe o *cluster* tem que ser vistos como clientes comuns.

A Figura 1 mostra um *cluster* formado por quatro computadores comuns. Eles estão interligados através de um *switch* e compartilham um dispositivo de armazenamento.
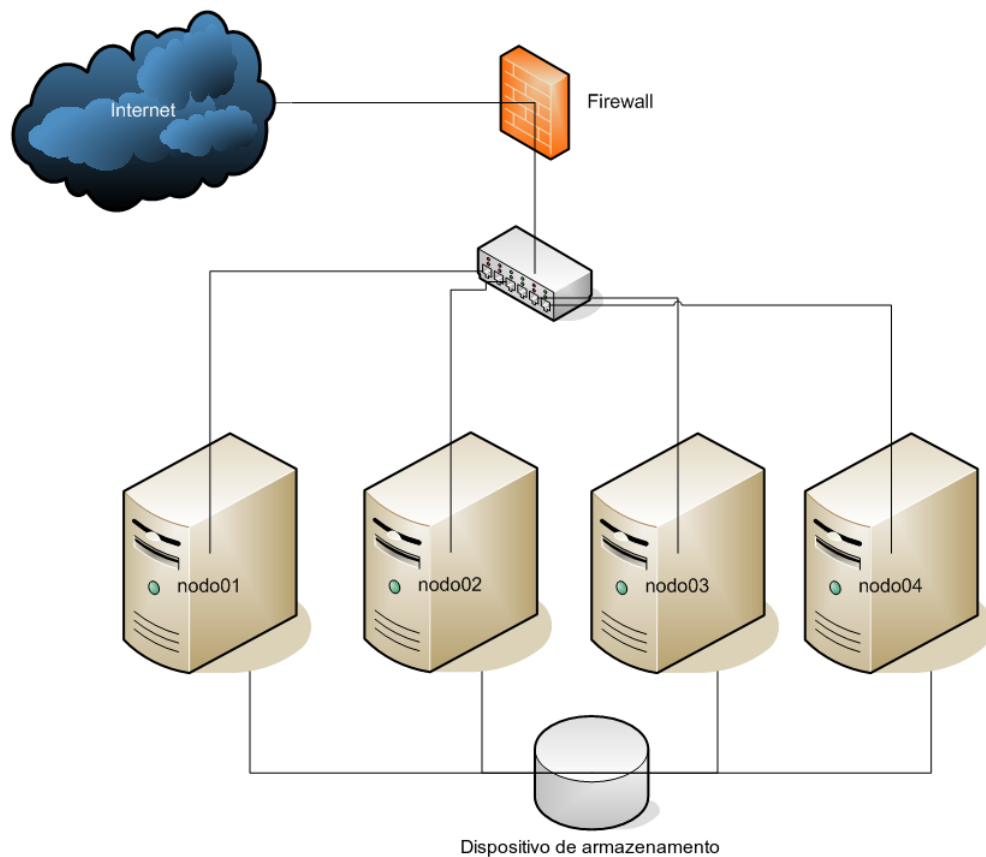
Figura 1: **Cluster de quatro computadores**

Fonte: O Autor (adaptado de Pereira, 2004 p. 6)

Estas são as características de um sistema de imagem única, ou seja, idependente do *cluster* ser composto por vários computadores interligados, toda a comunicação, distribuição de tarefas e sincronização de informações devem ser transparentes para o usuário. O *cluster* deve comportar-se como um sistema centralizado (FAUSTINO, 2006).

Redes de estações de trabalho não podem ser consideradas como *cluster*, pois cada estação trabalha de forma autônoma, o que não caracteriza um sistema de imagem única.

A utilização dos *clusters* permite que alguns objetivos sejam alcançados, tais como:

I. alto desempenho: por meio da interligação de computadores, tarefas são processadas paralelamente e conseqüentemente em menos tempo;

II. escalabilidade: capacidade de aumentar a capacidade de processamento do *cluster*, por meio da adição de novos nodos;

III. tolerância a falhas: aumento da confiabilidade do sistema, pois caso algum componente falhe outro assume sua função;

IV. baixo custo e independência de fornecedor: o custo de montagem do ambiente é baixo, pois é usado *hardware* comum que pode ser de qualquer fabricante, e utilizando *software* livre não existe a necessidade de pagamento de licença de uso de aplicativo.

2.3 ABSTRAÇÕES DE UM CLUSTER

Para melhor compreensão do trabalho é importante conhecer alguns jargões utilizados no mundo dos *clusters* e da alta disponibilidade. Esta seção apresenta os termos mais comuns, tais como nó, recurso, dependência de recursos e grupo de recursos.

Quando um programa é executado em um computador, este programa em execução é conhecido como processo. No sistema operacional Linux, um processo que disponibiliza um serviço é conhecido como *daemon*. Um *daemon* e os recursos oferecidos por ele são chamados de serviço (KOPPER, 2005).

2.3.1 Nó ou Nodo

Cada computador interligado ao *cluster* é conhecido como nó ou nodo. Esta interligação pode ser por meio de uma interface de rede ou de uma interface serial. Por meio dessa interligação é feita a troca de mensagens entre os nodos do *cluster*, e uma falha pode ser detectada na ausência dessas mensagens (VOGELS, 1998).

A tabela 1 apresenta os quatro principais componentes de um nodo

Tabela 1: **Os quatro principais componentes de um nodo**

Componente	**Descrição**
CPU	Componente de processamento principal, que lê e escreve na memória do computador
Memória	Armazenamento temporário de informações durante operações de entrada/saída
Repositório de armazenamento	Dispositivo que armazena informações. Geralmente um disco rígido
Interconexão	Canal de comunicação entre os nodos

Fonte: O Autor (adaptado de Pereira, 2004, p. 8)

2.3.2 Recurso

As funcionalidades oferecidas pelos nodos são conhecidas como recurso. Eles podem ser lógicos, como o nome de um servidor por exemplo, ou físicos, como um dispositivo de armazenamento. É fundamental o uso de uma ferramenta que possibilite o monitoramento dos recursos que fazem parte do ambiente, pois em caso de falha, esta ferramenta pode por exemplo, fazer com que o recurso seja disponibilizado em outro nodo e avise o operador que a falha ocorreu, de modo que possa ser executada uma ação corretiva no nodo que apresentou indisponibilidade (MISAGHI, 2006).

O processo de migração de um recurso de um nodo para outro, seja de forma automática por meio de intervenção de um operador, é conhecido como *failover* (PEREIRA, 2004).

2.3.3 Dependência de Recursos

Geralmente, um recurso depende de outro para funcionar. Por exemplo, o servidor web Apache depende de uma placa de rede e de um sistema de arquivos para poder funcionar. O sistema de arquivos para poder funcionar, precisa de um disco rígido, e assim por diante. Essa relação de dependência entre os recursos é conhecida como árvore de dependências, e descreve a ordem em que os recursos devem ser inicializados. Quando um nodo apresenta uma falha, é por meio da árvore de dependências que ele sabe quais recursos devem ser migrados em conjunto de um nodo para outro (VOGELS, 1998). A Figura 2 ilustra este exemplo.
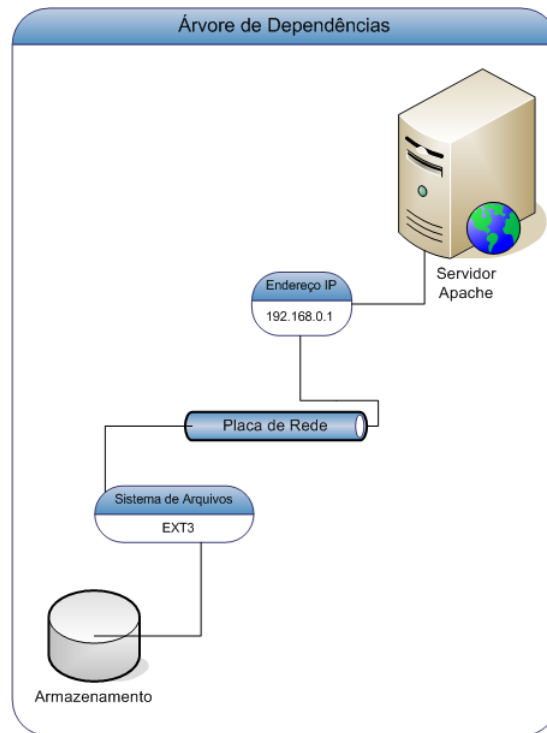
Figura 2: **Dependência de recursos em um servidor Apache**

Fonte: O Autor (adaptado de Pereira, 2004 p. 9)

2.3.4 Grupo de Recursos

Além da dependência de recursos, em que mais de um recurso pode ser migrado em conjunto, existe a possibilidade de se agrupar recursos de modo que em caso de falha, não apenas um, mas o grupo todo seja migrado.

Cabe ao administrador definir que um recurso só pode depender de outro que esteja no mesmo grupo (PEREIRA, 2004).

A Figura 3 mostra o caso 1, em que cada um dos recursos está separado. No caso 2, os recursos sistema de arquivos e endereço IP, foram agrupados ao Apache, ou seja, o Apache precisa deles para funcionar.

foi criado um grupo Apache ao qual os demais recursos estão agrupados.

2.4 A HISTÓRIA DOS CLUSTERS

Os *clusters* surgiram no momento em que uma tarefa não podia ser executada por um único computador e quando passaram a executar tarefas que exigiam um nível maior de confiabilidade, o que um único computador não poderia garantir. A data exata do surgimento dos
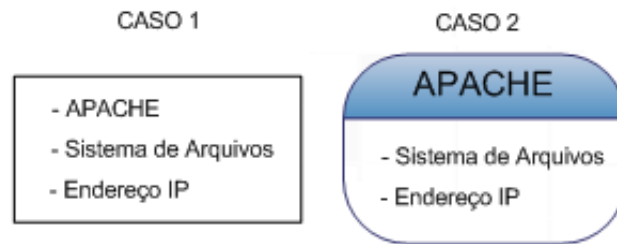
Figura 3: **Exemplo de agrupamento de recursos em grupo**

Fonte: O Autor

primeiros *clusters* é desconhecida, mas foi no final da década de 50 e início da década de 60 (MARCO, 2006).

Em 1967, em um artigo publicado por Gene Amdahl da IBM, é apresentada a *Lei de Amdahl*, por meio da qual é descrito matematicamente o ganho de velocidade ao se paralelizar tarefas seriais (AMDAHL, 1967).

Os *clusters* atuais tem uma relação forte com a evolução das redes de computadores, o desenvolvimento do protocolo Protocolo de Controle de Transmissão/Protocolo de Internet - *Transmission Control Protocol/Internet Protocol* (TCP/IP) e do sistema operacional Unix na década de 70. O conceito de redes baseadas na comutação de pacotes surgiu em 1962 e foi criado pela *RAND Corporation*. Usando o conceito criado pela RAND, a rede *ARPANET* surgiu em 1969 e foi um dos primeiros *clusters* criados usando *hardware* comum. Ela foi usada para criar a infraestrutura de interligação de quatro centros de computação nos Estados Unidos. Por volta de 1983, surgiram protocolos e ferramentas que permitiam fazer de maneira fácil a distribuição de trabalhos entre diferentes CPUs e também o compartilhamento de arquivos.

A primeira solução de *cluster* comercial surgiu em 1977. O *ARCNET*, desenvolvido pela empresa Datapoint, era uma solução composta por *hardware*, aplicativo e outros equipamentos que permitiram a interligação das máquinas em rede (WIKIPEDIA, 2006).

Em 1976 entrou no mercado a Tandem Computers, que por muitos anos dominou o mercado de soluções de tolerância a falhas baseadas em aplicativoṄo ano de 1980, foi fundada a Stratus Computers, que entrou no mercado fornecendo soluções de alta disponibilidade baseadas em *hardware*. Anos mais tarde, tanto a Tandem como Stratus acabaram se incorporando a outras empresas e atuando em ramos mais específicos (WEBER, 2002).

2.5 OS DIFERENTES TIPOS DE CLUSTER E SUAS APLICAÇÔES

Os *clusters* têm se mostrado uma alternativa viável se comparada a soluções proprietárias de alto desempenho e alta disponibilidade, pois os recursos financeiros exigidos na montagem do ambiente são menores. Alguns fatores impulsionaram a popularização dos *clusters*, como uso do *hardware* comum, queda de preços e evolução de processadores e equipamentos de rede (FAUSTINO, 2006).

Empresas do setor privado, órgãos governamentais, instituições de pesquisa ou instituições financeiras, utilizam *clusters* para resolver inúmeros tipos de problemas.

A bibliografia apresenta diversos tipos de *cluster*, em que a nomenclatura geralmente é baseada no tipo de aplicação que oferecem, tais como banco de dados ou armazenamento. Mas de modo geral, os três principais tipos são: *clusters* de alto desepenho, *clusters* de alta disponibilidade, e *clusters* de balanceamento de carga.

2.5.1 Clusters de alto desempenho - HPC

Neste tipo de *cluster*, tarefas que exigem um nível elevado de processamento são divididas entre os nodos, ou seja, são executadas de forma paralela, afim de que o tempo de processamento seja menor. Quanto mais nodos estiverem ligados ao *cluster*, menos tempo será necessário para executar todo o processamento (PEREIRA, 2004).

Clusters de alto desempenho podem ser empregados em ambientes que possuam uma grande demanda de dados a serem processados. Bancos de dados robustos, aplicações de engenharia, portais com grande volume de acessos são alguns exemplos.

A utilização de aplicações que fazem uso do processamento paralelo é uma tarefa complexa, por isso precisa ser bem planejada. Além do desenvolvimento das aplicações ser muito trabalhoso, exige grande quantidade de memória e bastante tempo de processamento.

No Brasil, existem cinco Centros Nacionais de Processamento de Alto Desempenho (CENAPAD) espalhados pelo país. Estão instalados junto à Unicamp em Campinas-SP, UFPE em Recife-PE, UFRGS em Porto Alegre-RS, UFMG em Belo Horizonte-MG e UFRJ, no Rio de Janeiro-RJ. Surgiram em 1994 com o objetivo de apoiar o avanço de pesquisas nas áreas de ciência e tecnologia no país. Para isso disponibilizam aos usuários laboratórios com equipamentos de alto desempenho, recursos de *hardware* e aplicativos, além do suporte técnico (CENAPAD, 2007). Também a Petrobrás utiliza *clusters* de alto desempenho nas plataformas,

para executar cálculos relacionados à extração de petróleo (FRANCO, 2004), e simulação numérica em reservatórios de petróleo (SCHIOZER, 1997).

2.5.2 Clusters de alta disponibilidade - HAC

Os Clusters de alta disponibilidade - *High Availability Clusters* (HAC) tem como objetivo manter um ou mais serviços no ar a maior parte do tempo possível e de forma segura (PITANGA, 2002). Os recursos têm que estar sempre, ou quase sempre, disponíveis para atender às requisições dos clientes, ou seja, têm de estar acessíveis por um período de tempo muito próximo a 100% (PEREIRA, 2004).

A bilbiografia mostra cálculos de disponibilidade tomando como base um ano, ou seja, o tempo de disponibilidade de um serviço durante o período de um ano é de um valor próximo a 100%, que pode variar de 99% a 99,999999999%, de acordo com a aplicação. Faustino (2006, p. 19) diz que:

> Manter apenas um computador realizando uma tarefa importante, não é garantia segura de que o serviço vai estar sempre disponível, pois problemas de *hardware* ou aplicativos podem causar a interrupção do serviço.

Assim, um dos requisitos para alcançar alta disponibilidade, é a eliminação dos pontos únicos de falha. A autora Weber (2002, p. 29) afirma que:

> Usuários que inicialmente se mostram satisfeitos em contar apenas com a simples automação de serviços, logo passam a desejar que esses serviços sejam prestados corretamente e sem interrupções. Sistemas tolerantes a falhas são caros e portanto empregados apenas naquelas situações em que sua não utilização acarretaria prejuízos irrecuperáveis.

A autora também cita algumas áreas em que os *clusters* de alta disponibilidade são empregados:

I. aplicações críticas de tempo real, como medicina, controle de processos e transportes aéreos;

II. aplicações seguras de tempo real, como transportes urbanos;

III. aplicações em sistemas de tempo real, de longo período de duração sem manutenção, como viagens espaciais ou satélites;

IV. aplicações técnicas, como telefonia e telecomunicações;

V. aplicações comerciais de alta disponibilidade, como sistemas de transação e servidores de rede (WEBER, 2002).

No Brasil, empresas como Telecomunicações Brasileiras S.A. (TELEBRAS) e Instituto Nacional de Pesquisas Espaciais (INPE) têm trabalhado na pesquisa de soluções tolerantes a falhas para o sistema de telefonia e espacial respectivamente.

Uma medida muito usada atualmente, é a do número de noves de disponibilidade.

2.5.3 Clusters de balanceamento de carga - LBC

Os Clusters de balanceamento de carga - *Load Balance Clusters* (LBC) são um misto de *cluster* de alto desempenho, com *cluster* de alta disponibilidade. Neste tipo de *cluster*, vários nodos, chamados de nodos diretores ou *load balancers*, atendem requisições de um serviço qualquer e repassam aos demais nodos do *cluster*. Estes por sua vez fazem o processamento das informações e são conhecidos como servidores reais (GARCIA, 2007). Esta técnica faz com que não haja um ponto de gargalo. É muito usada em grandes portais de Internet, que recebem um grande número de acessos simultâneos (ABREU, 2004).

Um exemplo de empresa que utiliza em grande escala os *clusters* do tipo LBC é o Google. Neste ambiente, diversos nodos recebem às requisições de busca feitas pelos usuários e repassam aos nodos reais, que então fazem a busca e devolvem os resultados ao usuário (DEAN, 2003).

2.6 ARQUITETURA DOS CLUSTERS

A arquitetura dos *clusters* compreende toda a estrutura de funcionamento dos nodos e equipamentos de rede, ou seja, a maneira com que eles se comunicam, quais serviços executam, o que fazer no caso de falhas, entre outros. Os elementos citados na seção 2.3 trabalhando em conjunto é que formam esta arquitetura.

Hoje não existe um padrão para arquitetura de *clusters*. Muitos fornecedores de soluções comerciais adotam uma nomenclatura voltada para suas soluções, e que diferem umas das outras devido aos diferentes tipos de *cluster*. Uma das propostas de padronização arquitetural dos *clusters* é o projeto OCF, que propõe o uso de uma camada de programação em comum entre todas as soluções de *cluster* (PEREIRA, 2004).

Em geral, a literatura apresenta várias propostas semelhantes. Pereira (2004, p. 9) divide esta arquitetura em grupo de camadas e grupo de serviços. O grupo de camadas é

composto pelas camadas de comunicação, ligação, integração e recuperação, que são definidas a seguir:

I. camada de comunicação: é a camada de comunicação em baixo nível entre os nodos do *cluster*. As interfaces de comunicação mais comuns são do tipo Ethernet ou serial e cada interface trabalha de maneira autônoma. Através desta camada são trocadas mensagens que verificam se um nodo está ativo ou não, e mensagens que verificam o estado do enlace. Também é por meio dessa camada que é realizado o sincronismo dos dados entre os nodos do *cluster*.

II. camada de ligação: esta é uma camada de alto nível construída sobre a camada de comunicação. Ela proporciona a criação de um enlace virtual, por meio da associação de todos os canais de ligação do nodo para este enlace vitual. O objetivo desta camada é tornar mais simples a troca de mensagens entre as camadas superiores, por meio da abstração.

III. camada de integração: durante o funcionamento do *cluster* pode ser necessário que ele passe por uma transição, ou seja, adicionar, remover ou subsubstituir nodo do grupo. Esta camada garante as alterações na formação do *cluster* e faz com que todos os nodos tomem conhecimento da alteração da composição do *cluster*.

IV. camada de recuperação: a função desta camada é recuperar-se das transições realizadas pela camada de integração. Ela transporta as variáveis do sistema e seus valores do nodo falho para o nodo ativo, deixando-o no mesmo estado do nodo falho antes deste apresentar problemas. Esta camada permite ainda desativar o acesso a certos serviços, até que a migração dos recursos para o outro nodo esteja completa.

O grupo de serviços é formado pelo serviço de quórum, serviço de barreiras, serviço de informações e serviço de nomes:

I. serviço de quórum: os *clusters* possuem a característica de poderem ser divididos em grupos menores, ou seja, um *subcluster* formado por um número menor de nodos. Essa divisão pode ser necessária por exemplo, em uma situação em que ocorre uma falha de comunicação em uma parte do *cluster*. Os nodos problemáticos são então separados de modo a não interferir no funcionamento dos demais.

   Dois *subclusters* não podem disponibilizar serviços simultâneamente, sempre deve existir um *subcluster* principal. Esta situação é conhecida como síndrome do cérebro dividido, ou em inglês, *split-brain syndrome* (KOPPER, 2005). Para evitar a síndrome do cérebro

dividido, cada nodo possui uma lista em que consta quais são os membros associados ao *subcluster*.

Esta camada decide qual dos *subclusters* possui quórum, ou seja, qual será o *subcluster* principal. Ela é responsável por fazer essa escolha de duas maneiras: votação ou recurso de quórum.

Na votação, o *subcluster* composto pelo maior número de nodos é eleito o principal. No recurso de quórum, o *subcluster* que possuir um certo recurso será escolhido como *subcluster* principal, não levando em consideração o número de nodos que o compõe.

Após concluída a etapa para eleger o *subcluster* principal, a lista dos *subclusters* é atualizada e informada aos demais nodos. Demais nodos que não pertencerem ao *subcluster* principal neste momento param de processar as informações.

A Figura 4 apresenta um *cluster* com seis nodos ativos. Em seguida dois deles falham e o *cluster* é dividido em dois *subclusters*. Os nodos falhos ficam isolados e os quatro nodos restantes assumem.
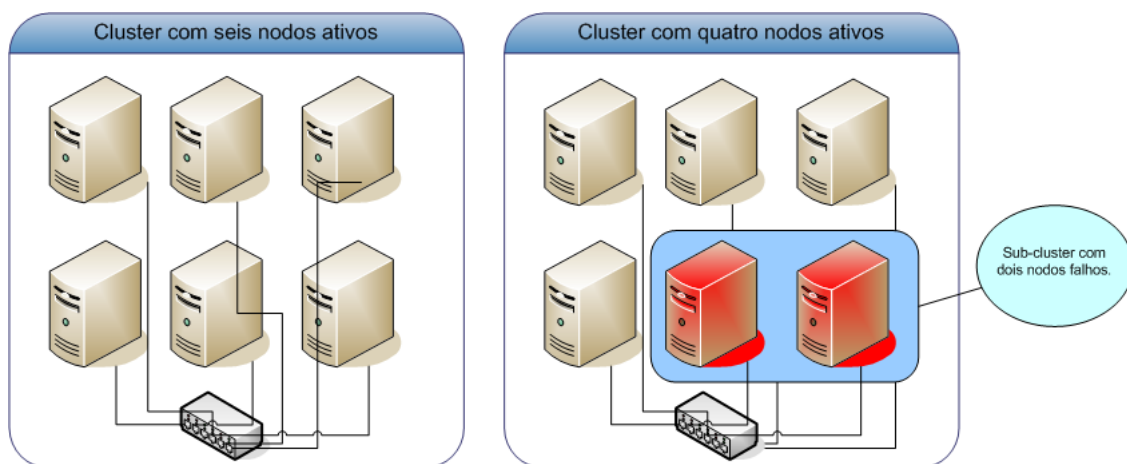
Figura 4: **Exemplo de cluster e sub-cluster**

Fonte: O Autor

II. serviço de informações: os nodos que compõe o *cluster* possuem capacidade de votar e tomar algumas decisões. Dessa forma, é necessário em cada nodo do *cluster*, um banco de dados transacional para que as configurações e informações sobre seu estado possam ser salvas de maneira persistente.

III. serviço de barreiras: esta camada controla todas as transições do *cluster*, ou seja, estabelece o sincronismo de processos entres os nodos. Para que exista o sincronismo são estabelecidas barreiras, as quais todos os nodos devem alcançar, para depois iniciar o processo de chegada na próxima. Se um nodo alcançar uma barreira antes dos demais,

ele aguarda até que todos os nodos cheguem no mesmo ponto, antes de dar sequência no processamento.

IV. serviço de nomes: é uma estrutura hierárquica através da qual um nodo pode registrar nomes e os processos podem consultar ou definir dependências baseadas nestes nomes.

2.7 CONCLUSÃO

Este capítulo apresentou os conceitos básicos relacionados aos *clusters*. Pode-se concluir que sistema operacional Linux oferece diversas vantagens em relação às soluções proprietárias de *cluster*, entre elas o fato de poder se utilizar *hardware* comum e utilizar aplicativos sem custo de licenciamento, que proporciona uma grande economia para as empresas e, de ter seu código fonte disponível, o que permite que empresas que tenham alguma necessidade muito específica, possam utilizar o código já disponível e adaptá-lo, criando desta forma uma solução para o seu problema.

Além disso, a proposta do OCF torna mais simples os processos de melhoria e desenvolvimento de novas ferramentas para uso nos ambientes de *clusters*, pois são desenvolvidas seguindo os mesmos conceitos, o que poupa trabalho e garanta uma maior compatibilidade entre estas soluções.

3 ALTA DISPONIBILIDADE

Os avanços alcançados nos últimos anos nas áreas de *hardware* e equipamentos para redes, as ações por parte do governo como a redução de taxas tributárias sobre computadores (MANFREDINI, 2005), são exemplos de ações que têm estimulado a queda de preços e a venda de equipamentos, fazendo com que cada vez mais, os computadores sejam empregados na realização de tarefas críticas e que as empresas fiquem cada vez mais dependentes deles. Exemplo disso, foi a grande preocupação com o *bug* do ano 2000. Além dos gastos financeiros, as empresas estavam preocupadas com a perda dos dados e, que isso pudesse causar um grande problema a nível mundial (VARGAS, 2000).

Um sistema de missão crítica, se falhar, pode trazer grandes prejuízos à empresa. Hoje, muitos processos dependem dos recursos computacionais para funcionar. Se os recursos não estiverem disponíveis, os processos param. Na área de comércio eletrônico, por exemplo, disponibilidade é fundamental, pois se um cliente não conseguir efetuar uma compra, ele se dirige ao portal do concorrente, pois a facilidade e o conforto proporcionados a ele são muito grandes (VARGAS, 2000).

Como as falhas não podem ser evitadas, um método de garantir a disponibilidade dos serviços, mesmo no caso de falha de um componente, é a redundância. Neste caso, os recursos falhos podem migrar para um equipamento operante. Este capítulo visa explorar os conceitos relacionados a alta disponibilidade através de exemplos, pois sua compreensão é fundamental para o entedimento do assunto.

3.1 OS PRINCÍPIOS DA ALTA DISPONIBILIDADE

Por meio de pesquisa feita na literatura, observa-se que muitas vezes são usados termos diferentes, ou até mesmo mais de um termo, para definir um mesmo conceito. Esta seção aborda de maneira detalhada os conceitos relacionados à alta disponibilidade.

Disponibilidade pode ser caracterizada pela porção de tempo em que um recurso está no ar e pode ser usado para a finalidade desejada (WATSON, 1995).

Para conceituar os termos relacionados à alta disponibilidade e facilitar a compreensão, ao longo do capítulo será utilizado um exemplo retirado e adaptado do portal do Instituto Nacional de Padrões e Tecnologia dos Estados Unidos - *National Institute of Standards and Technology* (NIST).

O exemplo do NIST considera uma ponte que fica sobre um rio. A ponte pode ser entendida como um recurso, que neste momento apresenta disponibilidade, pois permite o tráfego de veículos em ambos os sentidos. Desse modo, um caminhão que saiu de um lado do rio consegue fazer a travessia e chegar do outro lado sem problemas.

Redundância é a palavra chave para se alcançar alta disponibilidade e é usada desde os primeiros computadores, visando assim o aumento da confiabilidade do sistema (WEBER, 2002). Por isso, a eliminação dos Ponto Único de Falha - *Single Point of Failures* (SPOF)s é fundamental para o alcance da alta disponibilidade.

Sistemas que apresentam falhas fazem parte do dia-a-dia (WATSON, 1995). Segundo Pereira (2004, p. 38), tolerância à falhas é “quando um sistema pode mascarar a presença de falhas”. Tomando novamente o exemplo com a ponte, considerando que houve um acidente aéreo e que a ponte foi comprometida. Um avião chocou-se contra os pilares de sustentação e a estrutura toda veio a baixo. Se a ponte for o único meio de se atravessar o rio, pode-se entender que ela é um SPOF, pois neste caso não será mais possível fazer a travessia. Por outro lado, se existir outra ponte, ou uma balsa que permita fazer a travessia, pode-se dizer que o sistema apresenta redundância e é tolerante a falhas.

A Figura 5 mostra que a ponte do lado direito veio abaixo e como redundância a ponte do lado esquerdo está sendo usada, permitindo assim que o automóvel faça a travessia.
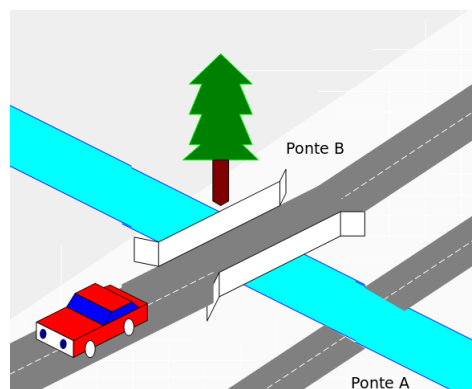
Figura 5: **Exemplo de redundância**

Fonte: O Autor

Quanto maior o nível de redundância entre os equipamentos envolvidos, maior será o nível de disponibilidade. Os SPOFs não se restringem somente aos recursos de máquina, mas também fonte de fornecimento de energia elétrica, ativos de rede e localização física dos equipamentos. Por exemplo, para evitar que um servidor fique indisponível em uma situação em que houver falta de luz, é importante que além de *nobreak*, possua duas fontes distintas de fornecimento de energia, tais como a rede elétrica convencional e um gerador de energia da

própria empresa. Assim no caso de uma fonte falhar, a outra assume a função. Outro exemplo, relacionado com a localização física dos recursos, é que se mantenha uma cópia de todos os dados da empresa em outro local físico, pois no caso de um incêndio, por exemplo, de nada adianta se ter redundância, ou *backup* das informações, se elas estiverem no mesmo local. Desse modo, a recuperação do desastre pode de dar de uma maneira bem rápida.

3.1.1 Como medir a disponibilidade

Para medir a disponibilidade de um recurso, atualmente existem duas maneiras que são mais comuns. A primeira considera a provável porção de tempo em que um sistema estará no ar e a porção de tempo em que estará fora do ar, conhecidas comumente como *uptime* e *downtime*, respectivamente (PEREIRA, 2004). A segunda e mais usada atualmente, considera o número de noves de disponibilidade que o recurso apresenta durante um ano. Pode ser calculada através da seguinte equação:

$$A = \frac{MTBF}{MTBF + MTTR}$$

Desse modo, um recurso com 4 noves de disponibilidade, significa que seu *uptime* é de 99,99%, ou seja, durante um ano, apresenta indisponibilidade durante cerca de 52 minutos, cerca de 1 minuto por semana.

A tabela 7 define as siglas usadas na equação:

Tabela 2: **Medidas de Confiabilidade**

Medida	**Significado**
MTBF - *mean time between failures*	tempo médio entre a ocorrência das falhas
MTTR - *mean time to repair*	tempo médio para reparo do sistema

Fonte: (Pereira, 2004 p. 18)

Com base na equação anteriormente apresentada, na tabela 3, (PEREIRA, 2004) mostra algumas medidas de disponibilidade, e na tabela 4 alguns sistemas e a disponibilidade por eles exigida.

Por exemplo, para computadores pessoais, o tempo que podem ficar fora do ar durante um ano, é de cerca de 36,5 dias. Já para sistemas de defesa militar, o tempo máximo aceitável é de 31,5 segundos.

Tabela 3: **Medidas de Disponibilidade**

Código	Uptime	Downtime	Downtime por ano	Downtime por semana
1	90 %	10%	36,5 dias	16 horas, 51 minutos
2	98 %	2%	7,3 dias	3 horas, 22 minutos
3	99 %	1%	3,65 dias	1 hora, 41 minutos
4	99.8 %	0.2%	17 horas, 30 minutos	20 minutos, 10 segundos
5	99.9 %	0.1%	8 horas, 45 minutos	10 minutos, 5 segundos
6	99.99 %	0.01%	52,5 minutos	1 minuto
7	99.999 %	0.001%	5,25 minutos	6 segundos
8	99.9999 %	0.0001%	31,5 segundos	0,6 segundos

Fonte: (Pereira, 2004 p. 19)

3.2 Dependabilidade

O objetivo da tolerância a falhas é alcançar a dependabilidade (WEBER, 2002). O termo caracteriza-se por entregar um serviço de maneira confiável, ou seja, quem interage com o sistema espera que ele funcione de maneira adequada.

A dependabilidade mede a qualidade de serviço e a confiança depositada nele. Alguns atributos podem ser compreendidos como pontos de vista da dependabilidade, como: disponibilidade, confiabilidade, confidencialidade, integridade, segurança e sustentabilidade.

I. a prontidão de uso leva à disponibilidade;

II. continuidade do serviço leva à confiabilidade;

III. o impedimento de revelações desautorizadas de informações leva à confidencialidade;

IV. a proteção contra alterações impróprias nas informações leva à integridade;

V. a proteção contra catástrofes ao ambiente e ao usuário leva à segurança;

VI. a possibilidade de submeter o sistema a reparos e evoluções leva à sustentabilidade.

Tabela 4: **Exemplos de sistemas e suas disponibilidades necessárias**

Código	Onde são exigidos
1	computadores pessoais, sistemas experimentais
3	sistemas de acesso
5	provedores de internet
6	CPD, sistemas de negócios
7	sistemas de telefonia, de saúde bancários
8	sistemas de defesa militar

Fonte: (Pereira, 2004 p. 19)

Diante disso, o termo dependabilidade mostra-se mais abrangente, se comparado simplesmente à disponibilidade.

3.2.1 Confiabilidade

A confiabilidade é uma função $R(t)$, que representa a probabilidade de um sistema estar operante até o instante *t*, e também uma medida de probabilidade, pois a ocorrência de falhas é um fenômeno aleatório (PEREIRA, 2004).

Segundo Weber (2002, p. 11), a confiabilidade implica algumas condições que são fundamentais: estado operacional no início do período, especificação, condições definidas e período de funcionamento.

I. estado operacional no início do período: é uma premissa básica, pois não é possível falar de confiabilidade em sistemas que já partem operando com defeito;

II. especificação: para saber se um sistema é confiável ou não, é necessário que se tenha uma especificação de como ele deve funcionar. Caso contrário não é possível saber se o sistema é confiável ou não. Tomando novamente o caso da ponte como exemplo, ela deve apresentar uma especificação de quantas toneladas de peso suporta, afim de evitar que um caminhão com uma carga muito pesada tente fazer a travessia a e ponte venha abaixo;

III. condições definidas: as condições como temperatura do ambiente ou nível de umidade do ar devem ser conhecidas, afim de que o sistema funcione de maneira adequada;

IV. período de funcionamento: o tempo em que o sistema vai estar no ar e funcional deve ser conhecido. No caso da ponte, supondo que ela seja elevadiça e possa ser usada somente das 6hs da manhã, até as 18hs. Deverá ser especificado então que no período das 18hs até as 6hs da manhã, ela estará indisponível não por ter ocorrido algum problema, mas por questões relacionadas a especificação.

A confiabilidade é a medida mais usada em sistema críticos, nos quais o reparo é impossível, ou sistemas em que curtos períodos de operação incorreta não são toleráveis.

3.2.2 Disponibilidade

A disponibilidade também é uma medida de probabilidade, dada pela função $A(t)$, que representa a probabilidade de um sistema operar corretamente num instante *t* (PEREIRA,

2004). Disponibilidade é o atributo mais usado e desejado em sistemas de missão crítica e não pode ser confundida com confiabilidade. Um sistema pode apresentar vários noves de disponibilidade, mesmo estando fora do ar em alguns momentos, enquanto está sendo reparado, desde que os períodos sejam curtos e não venham a comprometer a qualidade do serviço.
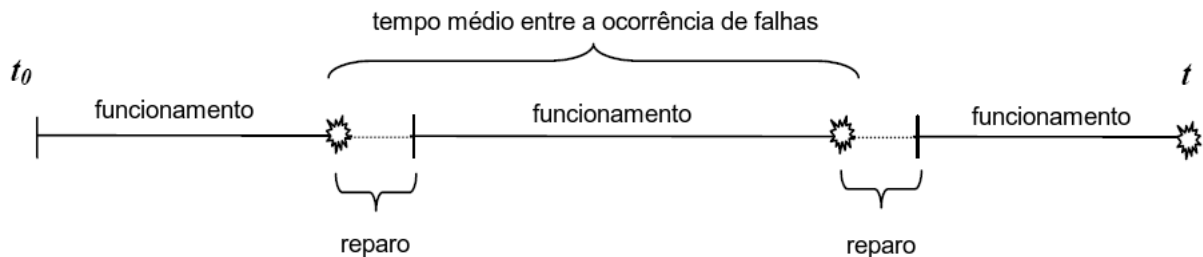
Figura 6: **Alternância entre os períodos de funcionamento e reparo**

Fonte: (Weber, 2002 p. 12)

3.2.3 Segurança

Por segurança, entende-se a capacidade de um sistema se comportar sem falhas, ou seja, operar dentro da sua especificação e, quando deixar de funcionar, não provocar danos aos sistemas que dele dependem (PEREIRA, 2004).

Tomando como exemplo o caso da ponte, é a garantia de que um veículo possa fazer a travessia do rio sem que ocorra algum problema e que quando a ponte for elevada, os veículos que dela dependem para fazer a travessia não sejam comprometidos.

3.3 CLASSIFICAÇÕES DOS CLUSTERS DE ALTA DISPONIBILIDADE

O autor (RESNICK, 1996) apresenta uma divisão quanto aos níveis de disponibilidade que podem ser alcançados. São eles: disponibilidade básica, alta disponibilidade e disponibilidade contínua.

3.3.1 Disponibilidade básica (basic availability - BA)

Um sistema que foi projetado, desenvolvido e implantado com um número suficiente de componentes (*hardware*, aplicativos e procedimentos), para satisfazer os requisitos funcionais e nada mais, é um sistema que pode-se dizer que apresenta Disponibilidade Básica (RESNICK, 1996).

Cabe aqui novamente o exemplo da ponte. Supondo que ela tenha sido construída tendo como objetivo somente permitir a travessia do rio, sem contar com nenhum recurso de redundância para o caso de a ponte precisar de reformas, então pode-se dizer que ela apresenta uma disponibilidade básica (RESNICK, 1996).

3.3.2 Alta disponibilidade (high availability - HA)

Um sistema que foi projetado, desenvolvido e implantado com componentes suficientes para satisfazer os requisitos funcionais e que também oferece redundância de componentes (*hardware*, aplicativos e procedimentos) para mascarar algumas falhas pré-estabelecidas, é um sistema que apresenta alta disponibilidade (RESNICK, 1996).

O mascaramento consiste em impedir que os usuários do *cluster* visualizem a falha, não impedir que ela aconteça. Isto pode ser alcançado através da redundância, através da substituição do equipamento falho pelo funcional. Dependendo do grau de transparência desejado, o grau de disponibilidade também pode variar. Os quatro tipos de mascaramento são: mascaramento manual, *cold stand-by*, *warm stand-by* e *hot stand-by*.

a) Mascaramento Manual - MM: Neste tipo de mascaramento, é necessária a intervenção de um operador para colocar o componente redundante no ar. Durante essa intervenção, o sistema fica indisponível para uso.

b) *Cold Stand-By* - CS: Após um componente falhar, os usuários são desconectados e perdem o trabalho feito até então. Neste momento ocorre o *takeover*, que é quando o nodo que estava somente aguardando a ocorrência de uma falha, entra no ar e assume o papel de nodo operante. A partir desse momento os usuários já podem conectar-se novamente ao serviço e terão disponíveis no servidor as informações do último sincronismo com o nodo principal.

c) *Warm Stand-By* - WS: Neste caso, os usuários também são desconectados e perdem o trabalho feito até o momento. Um mecanismo automático de detecção de falhas, detecta a falha e notifica o o nodo secundário, o qual já está iniciado e parcialmente ativo. Pelo fato de o nodo secundário já estar no ar, o tempo de detecção da falha é o mesmo do caso anterior, mas o tempo de migração de recursos de um nodo para outro é bastante reduzido.

d) *Hot Stand-By* / Replicação Ativa (HS)/(RA): Este tipo de mascaramento é o mais transparente para os usuários do *clusters* se comparado aos demais. Neste caso os componentes redundantes e ativos são logicamente agrupados, e assim são vistos como um recurso único pelo usuários. O estado dos processos é conhecido por todos os nodos do *clusters*. Após uma falha, os usuários são desconectados do nodo defeituoso, mas não observam erro algum,

pois os recursos continuam sendo oferecidos pelos nodos restantes e dessa maneira o trabalho continua podendo ser feito, ou seja, a recuperação é instantânea e não há nenhuma parada no funcionamento do sistema.

3.3.3 Disponibilidade contínua (continous availability - CA)

Os sistemas de alta disponibilidade somente mascaram as paradas não planejadas. A disponibilidade contínua trata também, as paradas planejadas dos serviços. Este tipo de modelo deve ser exclusivamente HS/RA para qualquer tipo de parada.

Para compreender melhor este problema, a Figura 7 apresenta um cenário com dois nodos, em que ambos precisam passar por uma atualização de aplicativo. Inicialmente, um processo é igualmente replicado entre os dois nodos. Para que seja feita a atualização, o nodo 2 é retirado de serviço, e recolocado com a versão do aplicativo atualizada (versão B). Quando ele entra no ar novamente, absorve o estado de processamento do nodo 1 com a versão A. O nodo 1 é então retirado de serviço e atualizado também. Finalmente o sistema entra no ar com a versão B instalada nos dois nodos. Para que essa atualização seja possível, tanto a versão A quanto a versão B, têm que ser suficientemente compatíveis a nível de compartilhamento de processos entre elas. Também a versão cliente tem que ser transparente para poder interagir com as versões A e B no servidor (RESNICK, 1996).
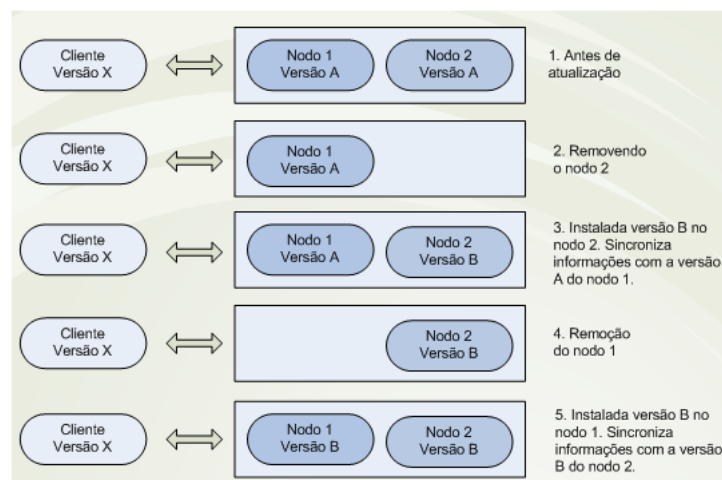
Figura 7: **Atualização de aplicativo em um sistema de disponibilidade contínua**

Fonte: O Autor (adaptado de Resnick, 1996 p. 10)

3.3.4 Domínios de disponibilidade

Em um sistema de alta disponibilidade estão envolvidos vários componentes, tais como *hardware*, aplicativo, procedimentos, entre outros. Geralmente não se deseja que todos os componentes ofereçam o mesmo nível de disponibilidade, ou seja, alguns componentes vitais para o funcionamento do sistema requerem um nível maior de disponibilidade, enquanto outros exigem um nível menor.

O autor (RESNICK, 1996), sugere uma divisão de seis sub-domínios, aos quais os componentes podem ser associados.
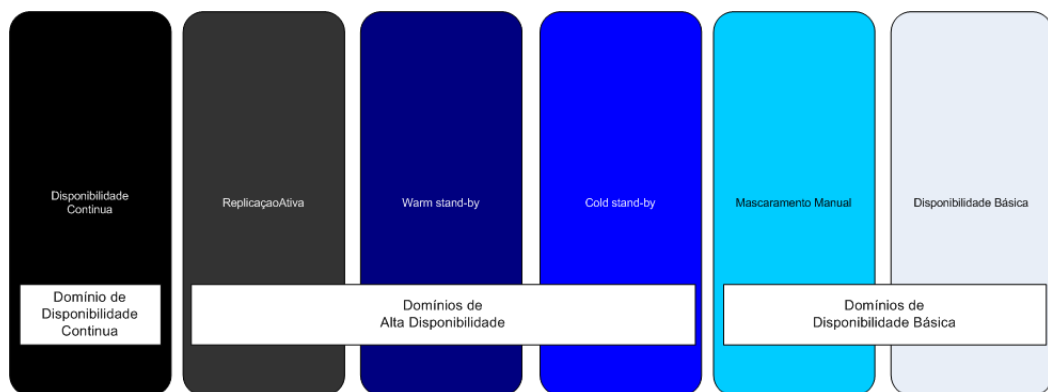
Figura 8: **Domínios de disponibilidade**

Fonte: O Autor (adaptado de Resnick, 1996 p. 11)

3.3.5 Replicação passiva e ativa

Os *clusters* permitem dois tipos de replicação de estados de processos: replicação ativa e replicação passiva.

a) replicação ativa - RA: o estado de todos os processos são compartilhados em tempo real entre nodos que compõe o *cluster* (BRASILEIRO, 2000). Eles são também coordenados, afim de que as transações ocorram de maneira sincronizada entre as réplicas (PEREIRA, 2004).

b) replicação passiva - RP: neste caso, os processos não são sincronizados em tempo real. Geralmente, a RP é empregada em ambientes onde não se deseja que os *clusters* sejam sincronizados por completo, ou em ambientes onde o custo de implementação da RA se torna muito alto.

3.4 DEFEITOS, ERROS E FALHAS

Ao usar um sistema, espera-se que ele funcione de acordo com sua especificação. O defeito (*failure*) ocorre quando o funcionamento foge dessa especificação (PEREIRA, 2004).

Um erro é o que pode levar o sistema ao estado de defeito (ALMEIDA, 2001).

O termo falha é usado para definir a causa física ou lógica que conduz ao erro (WEBER, 2002), ou seja, as deficiências no sistema que podem levar o sistema ao estado de erro.

Por ser uma propriedade do estado do sistema, o erro pode ser observado e avaliado. Por outro lado, o defeito não é uma propriedade do sistema, por isso pode ser detectado apenas quando o sistema apresentar algum erro.

Retomando o exemplo da ponte, citado no início do capítulo. Uma falha pode ser um buraco no acostamento da ponte. O fato do buraco estar presente conduz a ponte ao estado de erro, pois o buraco não faz parte do projeto da ponte. Enquanto o acostamento não for utilizado, o buraco não será responsável por nenhum acidente. Entretanto, no momento em que o acostamento for utilizado, um veículo pode cair no buraco e sofrer um acidente, conduzindo o sistema ao estado de defeito, pois o acidente também não faz parte da especificação da ponte.

3.4.1 O modelo de três universos

Este modelo mostra a relação entre defeitos, erros e falhas. É formado por três universos: universo físico, da informação e do usuário (SANTOS, 2000).

a) universo físico: é onde ocorrem as falhas. Ele é composto pelas partes físicas do sistema. Nele é que ocorrem as alterações de comportamento que levam a ocorrência de falhas.

b) universo da informação: é nele que os erros acontecem. Este universo é o responsável por afetar as informações armazenadas no computador.

c) universo do usuário: aqui o usuário percebe a ocorrência de erros, que é quando surgem os defeitos.

A representação gráfica do universo que envolve os três ítens comentados anteriormente é mostrada pela Figura 9:

Figura 9: **Modelo de três universos: defeito, erro e falha**

Fonte: O Autor (adaptado de Weber, 2002 p. 9)

3.4.2 A classificação das falhas

Segundo o autor Pereira (2004, p. 29), “as falhas podem ser classificadas de acordo com a natureza ou a duração”:

I. Natureza

a) falhas físicas: são as falhas relacionadas aos componentes físicos do sistema. Podem ocorrer devido a mudanças de temperatura, interferência eletro-magnéticas, curto circuitos, entre outros.

b) falhas humanas: podem ter ocorrido na fase de projeto do sistema, ou durante a interação do operador com o equipamento.

II. Duração

a) falhas transientes: falhas com curto período de duração. Elas também podem ser intermitentes, ocorrendo seqüencialmente por curtos intervalos de tempo.

b) falhas permanentes: uma vez que o componente falhou, ele não voltará mais a funcionar corretamente.

3.5 FASES DA TOLERÂNCIA A FALHAS

Não existe uma técnica que deve ser seguida em todos os casos em que se deseja alcançar alta disponibilidade. Porém existem alguns pontos que devem ser considerados e que levam a este objetivo, os quais serão detalhados no decorrer desta seção.

3.5.1 Detecção de erros

Antes dos erros serem tratados, eles precisam ser identificados. Portanto esta é a primeira fase da tolerância à falhas. Elas são descobertas através de erros apresentados pelo sistema, por isso é de extrema importância que os recursos sejam monitorados e verificados, de modo que as falhas possam ser corrigidas e o sistema opere conforme sua especificação (OYAMADA, 2003).

O ideal seria que todos os erros fossem detectados e tratados, mas isso acarretaria numa carga de trabalho muito grande e tornaria-se inviável. Portanto somente os componentes que podem causar erros de grande impacto devem ser considerados.

Segundo Pereira (2004, p.30), "algumas propriedades devem ser levadas em conta antes de se definir o que será verificado":

a) a verificação deve ser baseada somente na especificação do sistema. Ítens que fujam da especificação devem ser ignorados;

b) a verificação deve ser completa e correta, ou seja, todos os possíveis erros que possam ocorrer devem ser detectados e verificados:

c) a verificação deve ser independente do sistema que está sujeito a falhas, pois as verificações também podem falhar e, não se deseja que uma falha na verificação seja relacionada ao sistema que está sendo verificado.

O conhecimento prévio da implementação do sistema pode ser bastante útil na escolha das verificações a serem executadas. As verificações podem ocorrer de várias formas, dependendo do sistema e dos erros de interesse. Os principais tipos são: por replicação, temporizadas, estruturais, de coerência e de diagnósticos.

a) verificações por replicação: este tipo de verificação é muito comum e pode ser implementada sem que se tenha conhecimento de como o sistema funciona internamente. Ela consiste em replicar um componente do sistema, geralmente *hardware*, e comparar os resultados, afim de que se possa detectar os erros.

b) verificações temporizadas: se a especificação de um componente inclui restrições quanto ao tempo de resposta, este tipo de verificação pode ser aplicada. Neste caso é feita uma requisição e o tempo de resposta é comparado com o tempo descrito na especificação do sistema. Em *clusters* de alta disponibilidade a ausência da resposta pode indicar a falha de algum componente ou mesmo de um nodo. Em um *cluster* de balanceamento de carga por exemplo, o tempo de resposta muito alto pode indicar uma sobrecarga de trabalho.

c) verificações estruturais: para qualquer tipo de informação, dois tipos de verificação são possíveis, verificação semântica e verificação estrutural. A verificação semântica tenta conferir se o valor é consistente com o resto do sistema. A verificação estrutural somente garante que a estrutura dos dados é como deveria ser.

A forma mais comum de verificação estrutural é a codificação usada em *hardware*. Nela são aplicados *bits* extras aos dados, de forma que possa ser detectado se existe algum *bit* corrompido.

d) verificações de coerência: determinam se o estado de algum objeto no sistema está coerente, como por exemplo, se um determinado valor está dentro de um intervalo.

e) verificações de diagnósticos: neste caso, o sistema usa algumas verificações em seus componentes para determinar se ele está funcionando corretamente. A partir do conhecimento prévio de certos valores de entrada e respectivos valores de saída corretos, os valores são aplicados ao componente e e a saída é comparada com os resultados corretos.

3.5.2 Confinamento de estragos e avaliação

Se um sistema não estiver sendo monitorado constantemente, haverá um intervalo de tempo entre a ocorrência da falha e a detecção do erro. Durante este tempo o erro pode se propagar aos demais componentes do sistema. Por isso, antes de tomar medidas corretivas é importante que seja verificado com detalhes, qual o estado do sistema que está compromentido, ou seja, quais partes foram danificadas, o que ocorreu com elas e qual a causa.

3.5.3 Recuperação de erros

Assim que o erro for detectado e os detalhes citados no ítem anterior forem conhecidos, é necessário remover o erro do sistema. Isso pode ser feito de duas maneiras:

a) recuperação por retorno (*backward recovery*): neste modelo o sistema é retornado a um estado anterior, o qual se espera que esteja livre de erros. Para que isso possa ocorrer, existe um repositório estável onde periódicamente são criados pontos de controle *checkpoints*. Quando um erro é detectado no sistema, ele sofre um retorno *rollback* para o *checkpoint* funcional.

b) recuperação por avanço (*forward recovery*): ao contrário do método anterior, não existe nenhum *checkpoint* anterior que possa ser utilizado. É feita uma tentativa de avanço,

através da criação de um novo *checkpoint* livre de erros. O sucesso do novo *checkpoint* vai depender muito do estrago causado no sistema.

Na prática o primeiro método é mais utilizado, pois a restauração se dá partindo de um ponto que já estava operante anteriormente, ou seja, a garantia de funcionamento é maior. O segundo método é uma tentativa, a probabilidade de sucesso na utilização é menor e vai depender muito de como a aplicação trata os erros.

A tabela 5 mostra em resumo as técnicas apresentadas e a Figura 10 exibe a idéia por traz dessas técnicas:

Tabela 5: **Técnicas de recuperação**

Técnica	recuperação por retorno	recuperação por avanço
Definição	condução a um estado anterior	condução a um novo estado
Características	alto custo, mas de aplicação genérica	eficiente, mas danos precisam ser previstos
Implementação	pontos de controle	específica a cada sistema

Fonte: O Autor (adaptado de Pereira, 2004 p. 33)
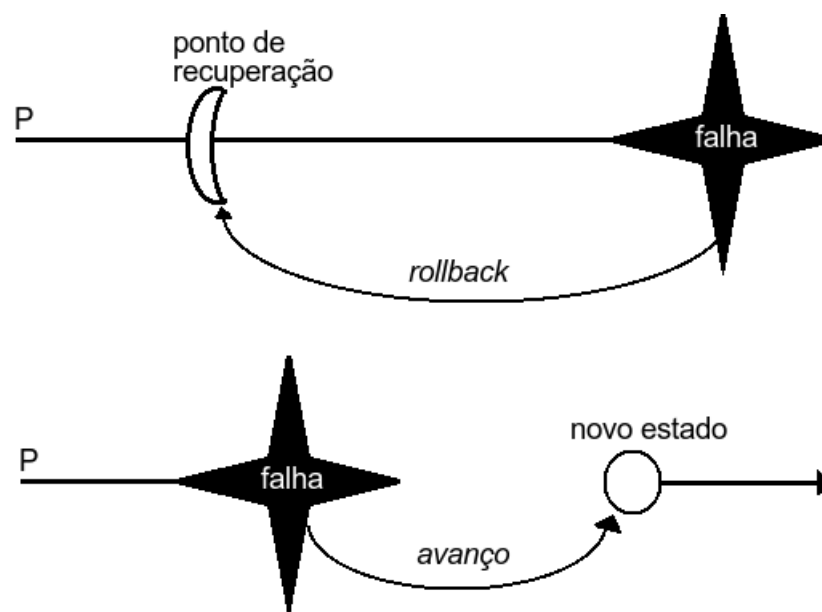
Figura 10: **Recuperação por retorno e por avanço**

Fonte: O Autor (adaptado de Pereira, 2004 p. 33)

3.5.4 Tratamento da falha e serviços continuados

Esta etapa descreve o tratamento das falhas, que visa impedir que novos erros aconteçam. A ocorrência de uma falha gera um erro, que podem ser gerados de duas maneiras:

I. erros gerados por falhas transenientes não precisam ser tratadas, pois já desapareceram;

II. erros gerados por falhas permanentes, ainda estão presentes no sistema mesmo após a recuperação. Mesmo que o sistema for reiniciado o erro ocorrerá novamente. Para evitar isso o componente defeituoso deve ser identificado e não ser mais utilizado. Se ele não for identificado, não será possível reparar o sistema de modo que o erro não aconteça novamente. Geralmente o componente defeituoso é o que está mais próximo da origem do erro. Em seguida, o sistema deve ser reparado, através da substituição do componente defeituoso para voltar a operar normalmente.

A tabela 6 apresenta alguns mecanismos que podem ser utilizados em cada fase da tolerância à falhas, e que auxiliam no processo de recuperação do sistema:

Tabela 6: **Fases da tolerância a falhas**

	Fases	**Mecanismos**
1	detecção de erros	duplicação e comparação; testes de consistência; diagnósticos
2	confinamento e avaliação	hierarquia de processos; controle de recursos
3	recuperação de erros	técnicas de recuperação por retorno ou avanço
4	tratamento da falha	diagnóstico e reparo

Fonte: (Pereira, 2004 p. 35)

3.6 CONCLUSÃO

Por tráz de toda a parte que envolve o *hardware* e aplicativos presentes em um ambiente de alta disponibilidade, existem as técnicas que tratam do comportamento destes componentes nas mais diversas situações. As técnicas apresentadas neste capítulo, se aplicadas corretamente e utilizadas em conjunto com boas ferramentas, permitem que se consiga tolerância a falhas. Isso proporciona o aumento na qualidade e confiabilidade de um sistema.

4 FERRAMENTAS PARA CRIAÇÃO DE UM AMBIENTE DE ALTA DISPONIBILIDADE

Este capítulo apresenta alguns aplicativos e ferramentas auxiliares que permitem a criação de um ambiente de alta disponibilidade. Serão enfatizados o Heartbeat, que permite que os recursos sejam migrados de um nodo falho para um nodo operante e o DRBD, que permite que uma partição de dados seja espelhada via rede.

4.1 COMO SURGIU O PROJETO LINUX-HA

O Heartbeat é um aplicativo que faz parte do projeto *Linux High-Availability* (LINUX-HA). O projeto LINUX-HA foi iniciado em 1997, pelo então funcionário da Bell Labs, Harald Milz. O projeto previa a criação de aplicativos que, permitissem montar um ambiente de alta disponibilidade usando o sistema operacional Linux, pois nativamente o Linux não possui estas características. Na época, Milz pesquisava o sistema operacional Linux como plataforma para desenvolvimento de aplicações e uma das suas tarefas era estudar a capacidade que o Linux tinha em trabalhar com aplicações de alta disponibilidade. Foi então que ele criou, e disponibilizou na Internet, um guia, contendo algumas recomendações que deveriam ser seguidas por quem quisesse desenvolver alguma aplicação voltada à alta disponibilidade. Depois de um ano, nada havia sido criado, então Milz decidiu implementar um pequeno aplicativo que era mencionado no guia. Assim nasceu o Heartbeat, que é a peça fundamental do projeto LINUX-HA (ROBERTSON, 2004).

A versão estável do Heartbeat para ambientes de produção foi disponibilizada em 1999, e desde então, vem sendo utilizado em inúmeros projetos pelo mundo todo, além de ter sido portado para outros sistemas operacionais baseados em Unix.

4.2 AS CARACTERÍSTICAS DO HEARTBEAT

O aplicativo Heartbeat foi criado inicialmente de modo a permitir que fosse montado um *cluster* de alta disponibilidade com dois nodos, um principal e outro secundário (backup) e que seguisse dois propósitos:

a) verificar se ambos os nodos estão no ar, ou seja, aptos a disponibilizar serviços aos clientes;

b) no caso do nodo principal falhar, o nodo secundário assumir e disponibilizar os serviços que eram oferecidos pelo nodo principal.

O Heartbeat utiliza três tipos de mensagens para troca de informações: *heartbeats*, mensagens de transição e requisições de transmissão.

I. *heartbeats*: são o tipo mais comum de mensagem. São conhecidas como batidas de coração, ou mensagens de *status* e têm de cerca de 150 *bytes*. É por meio delas que os nodos se comunicam. Na ausência destas mensagens é possível detectar a ocorrência de falha em um nodo. Elas usam o protocolo UDP e são enviadas por *broadcast*;

II. mensagens de transição: estas mensagens são do tipo mais raro. Elas contém toda a conversa entre o Heartbeat do nodo principal e o Heartbeat do nodo secundário no momento de transição, ou seja, quando os recursos estão sendo migrados de um nodo para outro;

III. requisições de transmissão: o Heartbeat usa uma numeração sequencial nos pacotes, afim de que eles cheguem corretamente. Isso faz com que o Heartbeat saiba quando os pacotes chegam corrompidos ou fora de ordem. Quando alguma coisa estiver errada, o Heartbeat solicita a retransmissão do pacote. Ele não faz isso mais do que uma vez por segundo, evitando assim o aumento de tráfego na rede (KOPPER, 2005).

O aplicativo Heartbeat, em execução no nodo backup, pode checar os *heartbeats* vindos do nodo primário através da interface Ethernet presente no servidor. Normalmente, o Heartbeat é configurado para utilizar uma conexão física separada da rede comum. Pode ser utilizada uma interface de rede dedicada ao Heartbeat, através de um cabo de cross-over por exemplo, ou pode ainda, ser utilizado um cabo serial. Se as três opções forem utilizadas em conjunto, eliminm o SPOF na comunicação entre os nodos.

O Heartbeat trabalhará sob uma ou mais dessas conexões simultâneamente e irá considerar que o nodo primário está ativo até que ele responda os *heartbeats* em pelo menos uma das interfaces (OYAMADA, 2003). A Figura 11 ilustra melhor esta situação através das três conexões entre os servidores:

A primeira versão do Heartbeat era muito simples, pois permitia somente a troca de *heatbeats* através de interface serial. As versões posteriores permitiram que o Heartbeat executasse *scripts* no nodo backup, quando o primário falhasse e, que, as mensagens de controle do Heartbeat entre os dois nodos também fosse feita através das interfaces de rede. A limitação de dois nodos não foi mais problema apartir da versão 2.0, que passou a suportar clusters com até 16 nodos (ROBERTSON, 2004).
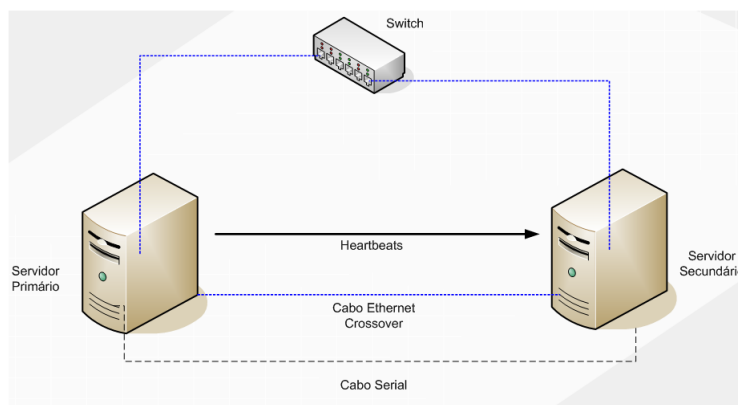
Figura 11: **Arquitetura do Heartbeat**

Fonte: O Autor (adaptado de Kopper, 2005 p. 112)

A comunicação via cabo serial é mais segura do que uma conexão *Ethernet*, pois através dela, o invasor não conseguirá abrir uma sessão *Telnet* ou *Secure Shell* (SSH), por exemplo. A limitação fica por conta do comprimento do cabo, que segundo a norma EIA-232, não pode passar de 15,24 metros, fazendo com que os servidores tenham que ficar no mesmo espaço físico (KOPPER, 2005).

A conexão *Ethernet* por sua vez, elimina os problemas com a distância. Ela permite também que o sistema de arquivos seja sincronizado entre os nodos do *clusters*. Usando dois caminhos físicos para conectar o nodo primário aos nodos secundários provê redundância para as mensagens de controle do Heartbeat, além de ser um dos requisitos para a eliminação dos pontos únicos de falha. Os dois caminhos físicos não precisam ser do mesmo tipo. Um *clusters* pode ser montado usando uma conexão *Ethernet* por exemplo, e uma serial.

Os serviços são disponibilizados em um IP que, é configurado em uma interface virtual, conforme a Figura 12. Sempre que os recursos são migrados de um nodo para outro, a configuração da interface virtual também o acompanha. Isso torna o processo de migração (*failover*) transparente para o usuário.

Os clientes que acessam os recursos do cluster, geralmente usam o protocolo ARP (Addess Resolution Protocol) para descobrir qual o endereço MAC (endereço físico) da placa de rede e o armazenam em uma tabela no sistema operacional, conhecida como tabela ARP. Porém, no momento em que ocorre o *failover*, o IP virtual do cluster passa a ter outro endereço MAC, neste caso, o endereço MAC da placa de rede do nodo secundário. Para garantir que os clientes acessem o servidor correto, o Heartbeat usa uma técnica conhecida como GARP *(Gratuitous ARP broadcasts)*. Essa técnica consiste em forçar a atualização da tabela ARP nos clientes através de *broadcast* (KOPPER, 2005).
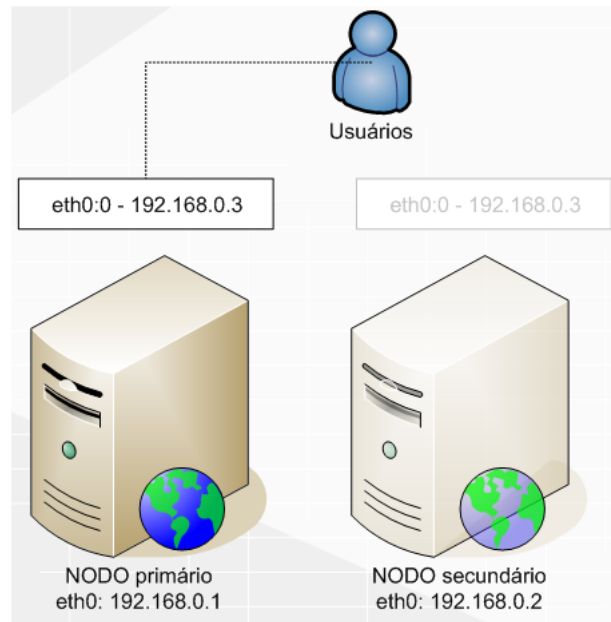
Figura 12: **Uso da interface virtual no cluster**

Fonte: O Autor

Nas últimas versões, o Heartbeat ganhou também um mecanismo para tirar do ar outros nodos, conhecido como Atire na Cabeça do Outro nodo - *Shot The Other Node In The Head* (STONITH), o qual provê um mecanismo de acesso único ao *storage* (recurso de armazenamento). Isso permite ao Heartbeat, trabalhar com dispositivos de armazenamento compartilhados. como o DRBD, por exemplo, garantindo assim a confiabilidade dos dados (KOPPER, 2005).

A partir desse momento o Heartbeat atingiu um alto grau de maturidade e passou a atrair muitas empresas da área de telecomunicações. Como resultado, o projeto passou a receber contribuição e apoio de algumas dessas empresas, em especial a Intel, devido ao seu interesse na área de telecomunicações.

4.2.1 A estrutura do aplicativo

As configurações do Heartbeat estão divididas em três arquivos: `/etc/hd.d/ha.cf`, `/etc/ha.d/authkeys` e `/etc/hd.d/haresources`.

`/etc/hd.d/ha.cf` é o a arquivo de configuração geral do Heartbeat. Nele são configuradas, por exemplo, quais interfaces serão usadas para os nodos se comunicarem (serial, *Ethernet*) e, quanto tempo o nodo secundário deve esperar antes de assumir, no caso do nodo principal falhar.

`/etc/ha.d/authkeys` permite assinar digitalmente os pacotes do Heartbeat. Podem ser usados três algoritmos:

I. MD5: O *Message-Digest algorithm 5* é definido pela RFC 1321. Sua chave, de 128 bits, é composta por 32 caracteres no formato hexadecimal.

II. SHA1: conhecido também como *Secure Hash Algorithm 1*. É definido pela RFC 2104 e usa uma chave de 160 bits para assinar os pacotes. Aos poucos o SHA-1 está substituindo o MD5, por proporcionar uma criptografia mais forte.

III. crc - é o metodo mais inseguro, pois não usa criptografia. Geralmente nos clusters de alta disponibilidade, é usado para fazer checar e detectar alterações que possam ter ocorrido nos pacotes ao longo do caminho.

A Figura 13 mostra a sintaxe para os três algoritmos. O arquivo é composto por duas linhas. A primeira deve conter a palavra *auth*, seguida de um número. A segunda linha deve conter este mesmo número seguida do algoritmo e de uma senha. A exceção fica por conta do algoritmo CRC, que por não usar criptografia, não necessida de senha.

```
[marcelo@canoeh ha.d]$ cat authkeys
# Algoritmo CRC
auth 1
1 crc

# Algoritmo SHA-1
auth 1
1 sha1 senha

# Algoritmo MD5
auth 1
1 md5 senha
```

Figura 13: **O arquivo de configuração `/etc/hd.d/authkeys`**

Fonte: O Autor

`/etc/hd.d/haresources` especifica qual será o nodo primário e quais recursos ele estará disponibilizando. A Figura 14 segue um exemplo do arquivo de configuração:

```
[marcelo@canoeh ha.d]$ cat haresources
nodo01.itflex.local     IPaddr2::192.168.200.203/24/eth0:0
nodo01.itflex.local     drbddisk::r0 Filesystem::/dev/drbd0::/rede::ext3
nodo01.itflex.local     smb
```

Figura 14: **O arquivo de configuração `/etc/ha.d/haresources`**

Fonte: O Autor

A sintaxe do arquivo é simples. No lado esquerdo deve ser configurado um nome válido para o nodo, ou seja, é necessário que um endereço IP responda por este nome. Para isso pode ser configurado um serviço de DNS, ou usado o arquivo `/etc/hosts`. Este nome não precisa, necessariamente, ser o mesmo no qual são disponibilizados os serviços e, precisa ser válido somente para o cluster, os usuários não precisam ter acesso a ele. Por isso, e por ser mais simples, a opção do arquivo `/etc/hosts` é a mais usada.

Do lado direito é configurado o recurso que ele está disponibilizando e que pode ser migrado para outro nodo no caso de falha, que neste exemplo são o endereço IP virtual, a partição do DRBD e o servidor Samba. O Heartbeat segue a ordem do arquivo `/etc/hd.d/haresources` para inicializar os serviços, ou seja, quem estiver na primeira linha é iniciado primeiro, quem estiver na segunda linha é iniciado em seguida e assim por diante.

A Interface para Programação de Aplicações - *Application Programing Interface* (API) do Heartbeat foi criada seguindo o padrão Base Padrão do Linux - *Linux Standard Base* (LSB). Isso permite que ele seja usado em distribuições Linux que sigam este padrão sem problemas, além de permitir que sejam facilmente criados plugins para trabalhar em conjunto com ele.

Segundo o padrão LSB, os serviços disponibilizados pelo sistema operacional devem ser controlados (iniciados ou parados) através dos *scripts* conhecidos como *init scripts* (DARWIN, 2006). Nas distribuições baseadas no RedHat Linux, estes arquivos estão localizados em `/etc/rc.d/init.d` e estão ligados ao nome do serviço, por exemplo, `smb`, `sshd` ou `cups`. Normalmente, o próprio sistema operacional trata de iniciar estes serviços no momento em que o servidor é ligado. Quando um serviço for disponibilizado no cluster, a tarefa de iniciar e parar o serviço deve ficar por conta do Heartbeat e não do sistema operacional. Isso fará com que o Heartbeat execute corretamente o processo de *failover*.

O Heartbeat usa a saída do comando, que é ecoada na tela para saber se o serviço foi inicializado com sucesso ou não. Os *init scripts* das distribuições Linux RedHat e SuSE, por exemplo, imprimem na tela o *status*, como *OK*, *done*, ou *sucess*.

```
[root@canoeh ~]# service smb start
Iniciando serviços SMB:                                    [  OK  ]
Iniciando serviços NMB:                                    [  OK  ]
[root@canoeh ~]#
```

Figura 15: **Serviço samba sendo iniciado no RedHat Linux**

Fonte: O Autor

4.2.2 Outras características do Heartbeat

Além das capacidades já apresentadas, o autor Robertson (2004, p. 07) apresenta outras características do Heartbeat.

O Heartbeat permite trabalhar com configurações no modo ativo/passivo ou ativo/ativo. Em um ambiente com dois nodos, se um disponibiliza todos os serviços e o outro atua como servidor backup, isso é chamado de configuração ativo/passivo. Se os dois estão configurados para prover serviços de alta disponibilidade e cada um atua fazendo *failback* (recuperação da falha) para o serviço falho do outro, isso é conhecido como configuração ativo/ativo. Em um ambiente ativo/passivo, o Heartbeat utiliza a função de *iptakeover*, para migrar de um nodo para outro o IP virtual. Antes este processo era executado por um *script* separado. Nas versões mais recentes o próprio Heartbeat executa este processo.

O Heartbeat permite que o administrador escolha como a recuperação de falha será tratada. Se a opção de *autofailback* estiver ativada, os recursos migrarão automaticamente para o nodo principal quando ele estiver disponível. Esta opção deve ser usada caso o *clusters* seja do tipo ativo/ativo. O Heartbeat permite que o administrador seja notificado por email quando ocorrer o *failover* ou *failback* no *clusters*.

O Heartbeat implementa através de plugins, comunicação sobre diferentes tipos de mídia, que atualmente inclui unicat, multicast e broadcast, além da comunicação através de porta serial. Não há limite quanto ao número de diferentes tipos de meios de comunicação que podem ser usados.

O Heartbeat nativamente não tem dispõe de um módulo que permita o monitoramento de recursos nos nodos do *clusters*. Isto pode ser feito através de um *watchdog*, que é um módulo baseado em *hardware* ou aplicativo que tem como função iniciar novamente um serviço, caso ele fique indisponível. Um dos fatores mais importantes que os *clusters* de alta disponibilidade oferecem em relação aos demais, é o tempo de recuperação de falhas (MTTR). Para um *clusters* de alta disponibilidade, o MTTR é significativamente influenciado pelo tempo de detecção de falha de um nodo membro do *clusters*. A maioria dos clusters de alta disponibilidade detectam falhas entre 10 e 60 segundos. O Heartbeat pode ser configurado para detectar falhas em menos de um segundo. Este tempo é crítico para certas aplicações, como na área de telefonia por exemplo.

Por fim, o Heartbeat oferece suporte básico ao OCF a partir da versão 2.0 do Heartbeat, que foi construída baseando-se na API do OCF.

Para as futuras versões, estão previstas melhorias na documentação. Apesar dela ser bastante completa, seu entendimento e a decisão de como utilizar o aplicativo em alguns casos é um pouco complicado. A idéia é disponibilizar uma documentação mais clara e com um aspecto mais profissional.

Além disso, o aplicativo pretende suportar novas mídias de comunicação. Atualmente as mensagens de controle do Heartbeat usam a comunicação *Ethernet* ou serial. As novas versões pretendem usar novos meios de comunicação, como USB por exemplo.

4.3 VISÃO GERAL SOBRE O DRBD

O Dispositivo de bloco de armazenamento distribuído - *Distributed Replicated Block Device* (DRBD) foi criado especialmente para uso em clusters de alta disponibilidade. Ele permite o espelhamento de dados entre duas ou mais partições, semelhante ao espelhamento conhecido como RAID-1. A diferença é que no RAID-1 os dados são espelhados em dois discos localizados fisicamente na mesma máquina (TESTONI, 2005). No DRBD esse espelhamento é feito entre dois ou mais nodos interligados em rede (KOPPER, 2005). A Figura 16 ilustra os dois casos.
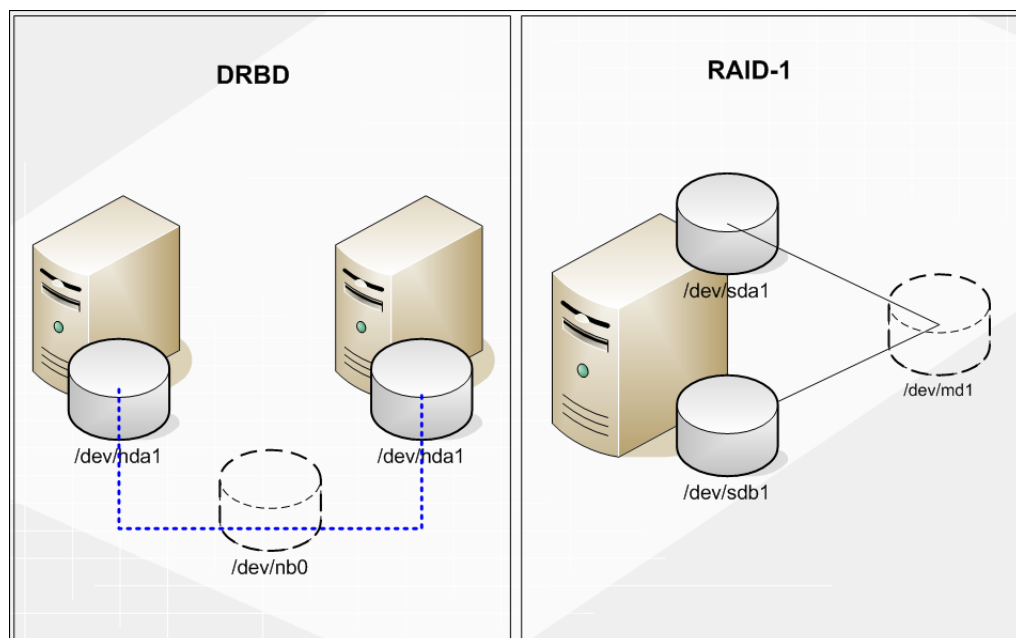
Figura 16: **Semelhança entre DRBD e RAID-1**

Fonte: O Autor

A nível de sistema operacional, o DRBD é um módulo do *kernel* do Linux que, juntamente com alguns *scripts* e aplicativos, permite que partições possam ser espelhadas.

Cada partição envolvida no espelhamento tem um estado, que pode ser primário ou secundário. O DRBD cria, em todos os nós, um vínculo entre um dispositivo virtual (/dev/nbX) e uma partição local (/dev/hdaX), a qual não é acessada diretamente, ou seja, toda a escrita é realizada no dispositivo virtual do servidor primário, que então transfere os dados para o dispositivo de bloco do nível mais baixo, a partição, e propaga aos demais nodos, em estado secundário. Ao receber os dados, o nodo secundário simplesmente escreve os dados no dispositivo de bloco do nível mais baixo.

Atualmente a versão 0.8 do DRBD permite que dois nodos acessem um dispositivo no modo leitura/escrita, simultâneamente. Para isso, é necessário utilizar um sistema de arquivos compartilhado, tal como Sistema de Arquivos Global - *Global File System* (GFS) ou Sistema de Arquivos Oracle versão 2 - *Oracle Cluster File System version 2* (OCFS2), por exemplo. Eles são capazes de gerenciar a trava, ou seja, permitir acesso exclusivo ao arquivo de maneira global ao sistema de arquivos, garantindo assim a integridade dos dados. Sistemas de arquivos comuns, tais como ext3, reiserfs ou xfs não devem ser usados com o DRBD neste tipo de operação.

Porém, apenas um nodo por vez fazendo o acesso leitura/escrita já é suficiente para o *failover* usual de um *clusters* de alta disponibilidade. Apesar do módulo do DRBD já fazer parte de algumas distribuições do sistema operacional Linux nativamente, geralmente o pacote DRBD precisa ser obtido na internet e deve ser compilado e instalado a parte, pois originalmente seu módulo DRBD não faz parte do *kernel* padrão do sistema operacional Linux.

O DRBD usa a mesma semântica de um dispositivo compartilhado, mas ele não necessita de nenhum *hardware* incomum, por trabalhar sobre redes que usam o protocolo TCP/IP, as quais têm um custo de implementação menor, se comparado com sistemas de armazenamento especial. Isso permite, por exemplo, que o DRBD seja usado em redes geograficamente afastadas (MUSSI, 2007).

Se o nodo primário falhar, o Heartbeat permutará o dispositivo secundário em primário e iniciará a aplicação ali.

Se o nodo que falhou retornar, ele se torna o secundário e precisa sincronizar seus dados com o primário. Isto é feito em segundo plano, sem que seja necessária a interrupção do serviço.

4.3.1 A estrutura do aplicativo

Os principais componentes do pacote DRBD são:

a) `módulo DRBD`: é um módulo do *kernel* do sistema operacional Linux, responsável pela interação com o *hardware* do nodo principal e nodos secundários.

b) `/sbin/drbdsetup`: é o comando utilizado para a configuração do DRBD. Através dele são criados os dispositivos virtuais (/dev/drbdX) e definidos quais nodos farão parte do *cluster*, por exemplo.

c) `/sbin/drbdadm`: comando utilizado para administração dos dispositivos DRBD. Ele permite que o espelhamento seja iniciado, parado, além de fornecer estatísticas sobre o espelhamento.

d) `/etc/drbd.conf`: arquivo onde são armazenadas todas as configurações do aplicativo.

4.3.2 As topologias de armazenamento

O DRBD permite ser utilizado em conjunto com algumas topologias de armazenamento existentes: armazenamento externo e armazenamento interno e distribuido.

a) armazenamento externo: Este tipo de sistema de *storage* (armazenamento), permite acesso simultâneo aos dados por todos os nodos que compõe o *cluster*. Esta solução centraliza todo o sistema de armazenamento dos dados e centraliza a gestão do sistema de arquivos em um único ponto, eliminando o trabalho de espelhamento dos dados pelo gestor do *clusters*. Além disso esta solução permite um melhor aproveitamento do espaço disponível. A grande desvantagem é que os sistemas de *storage* têm um custo muito elevado (LAUREANDO, 2004).

b) armazenamento interno e distribuído: Esta é a topologia na qual o DRBD se encaixa. Nela os nodos usam discos comuns (HDs), os quais possuem pelo menos uma partição de dados que é compartilhada no *clusters*. A grande vantagem desta topologia é a eliminação de SPOFs, pois se um disco em um nodo falhar, outro pode assumir imediatamente.

No caso do *storage*, se ele falhar, somente tendo outro no lugar para substituir e, por ser um equipamento caro, geralmente as empresas não tem. Levando em conta custos e manutenção, o espelhamento é a opção mais viável (REISNER, 2002).

4.3.3 Os protocolos utilizados pelo DRBD

No momento em que o DRBD foi projetado, foram considerados vários tipos de ambientes onde aplicativo poderia ser utilizado. Por isso, foram criados três protocolos para controle do

espelhamento dos dados nos nodos secundários. São eles: protocolo A, protocolo B e protocolo C.

I. protocolo A: o nodo principal executa uma operação de entrada/saída no disco local, notifica aos demais nodos sobre o que foi alterado e dá como encerrada a operação. O protocolo A não aguarda o recebimento de uma confirmação, nem se preocupa em saber se a operação foi ou não bem sucedida. Geralmente é usado em redes onde a latência é grande, como por exemplo uma conexão via Rede Virtual Privada - *Virtual Private Network* (VPN). Do ponto de vista da confiabilidade, é o que apresenta a menor nível entre os três.

II. protocolo B: apresenta um nível de confiabilidade um pouco maior. Após o nodo principal ter executado uma operação de entrada/saída no disco local, notifica os demais e aguarda a confirmação de que os pacotes foram recebidos, porém não aguarda que os nodos secundários façam a gravação nos discos locais (GARCIA, 2007).

III. protocolo C: as operações são realizadas de modo sincronizado. O nodo principal não sinaliza como finalizada uma operação de entrada/saída no disco local por exemplo, até até que todos os nodos secundários tenham executado a tarefa também e tenham enviado uma confirmação para o nodo principal.

Os desenvolvedores recomendam o uso do protocolo C em redes de baixa latência e que tenham um alto número de operações de entrada/saída, como um servidor de arquivos por exemplo. O emprego do protocolo C em redes de grade latência é totalmente inviável, pois causará uma grande lentidão no servidor. O protocolo C é o que proporciona maior segurança e confiabilidade entre os três, sendo portanto o mais utilizado (REISNER, 2004).

4.3.4 O sincronismo quando um nodo entra no *cluster*

Quando um nodo falho retorna ao *clusters*, ou quando um novo nodo é adicionado ao *clusters*, é necessário que seja feita a sincronização dos discos. Todos os nodos devem estar com os discos sincronizados em relação ao principal, pois isso possibilitará que um deles assuma caso o principal falhe.

No DRBD, o processo de sincronização foi implementado de modo a não afetar o desempenho do *clusters*. A operação é executada em segundo plano e permite que o administrador estabeleça qual a largura de banda máxima que pode ser usada na rede para o processo de sincronismo (REISNER, 2002).

Nas primeiras versões do DRBD sempre que um nodo voltava a fazer parte do *clusters*, era feito um sincronismo completo entre os dispositivos DRBD. Isso era problemático, pois du-

rante o período em que o *clusters* fazia o sincronismo, se o principal falhasse, os dados no nodo secundário estariam inconsistentes. Nas novas versões, o sincronismo é implementado através de um algoritmo conhecido como *active-logging*,

O sincronismo geralmente é feito através de uma conexão de rede dedicada ao DRBD. Uma conexão *gigabit* entre os nodos do *clusters* é uma boa opção neste caso. Isso evita o congestionamento na interface de rede dedicada aos usuários.

Se um nodo entra no *clusters* pela primeira vez, todos os blocos precisam ser copiados para os nodos secundários, que estão no modo de espera. Esse processo é conhecido como sincronismo completo.

Se um nodo apenas deixa o *clusters* por um certo período de tempo, é necessário que seja copiado somente as informações que foram alteradas no período de tempo em que o nodo esteve ausente. Este processo é conhecido como sincronismo rápido.

4.4 CONCLUSÃO

Com base na pesquisa feita pode-se identificar alguns aplicativos que podem ser utilizados em conjunto com o sistema operacional Linux e que proporcionam alta disponibilidade. Alguns projetos tiveram um período de vida muito curto, por possuírem características que não vão de encontro com as necessidades do mercado. O Heartbeat e o DRBD foram os aplicativos de maior destaque, e por isso foram frutos de estudo.

Ambos os projetos amadureceram no decorrer do tempo, sendo hoje estáveis e podendo ser utilizados para solucionar os mais diversos tipos de problema.

Algumas funcionalidades ainda são requeridas. Muitas já estão previstas para as próximas versões, o que é garantia de que os aplicativos continuarão sendo desenvolvidos, e com estas novas características poderão ser ainda empregados em diversas outras soluções.

5 AVALIAÇÃO DAS FERRAMENTAS DE ALTA DISPONIBILIDADE

Para que se pudesse compreender melhor o funcionamento das ferramentas apresentadas no capítulo 4, foi montando um *cluster*de alta disponibilidade com dois nodos. Através dele, foi possível passar por situações semelhantes a um ambiente real, que tratam desde a instalação do sistema operacional, configuração básica do sistema, configuração dos aplicativos de alta disponibilidade, até o uso do *cluster* depois de tudo pronto.

5.1 A montagem do ambiente

Para a montagem do cluster, foram usados dois computadores com as mesmas configurações. Cada um tem seu endereço IP real usado para a comunicação dentro do *cluster*. Além disso, o Heartbeat controla outro endereço IP, neste caso `192.168.200.203`, que á atribuído ao nodo que está ativo no momento, através de uma interface de rede virtual, chamada de `eth0:0`. É por meio deste endereço também que os usuários acessam os recursos disponibilizados no *cluster*.
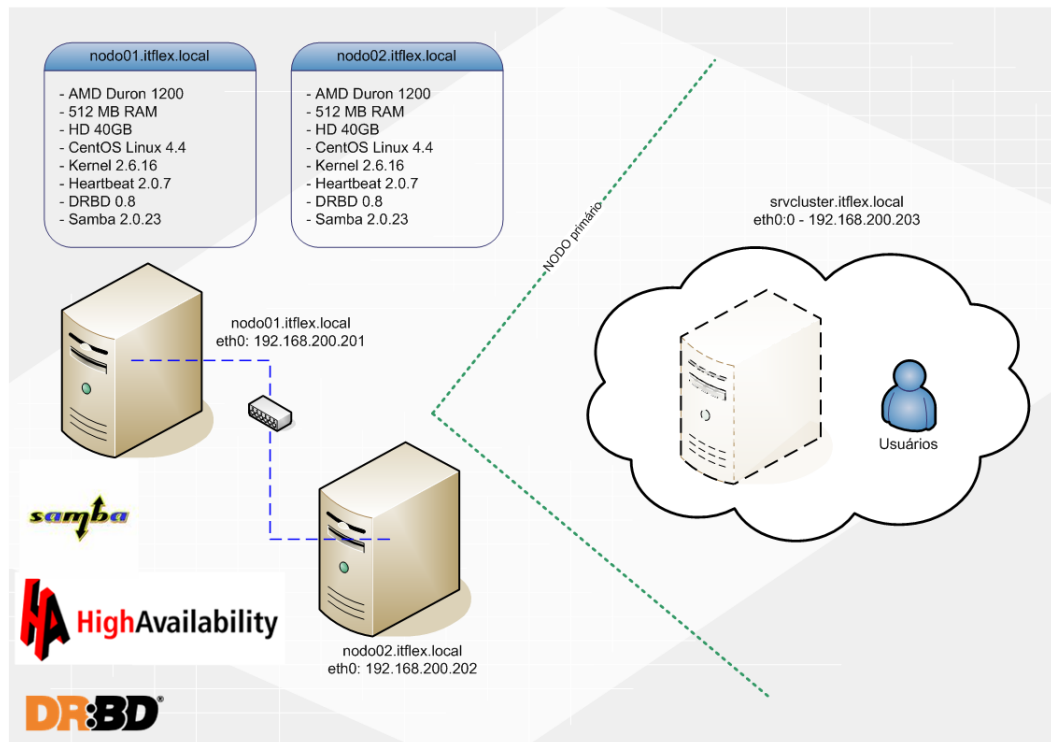
Figura 17: **Cluster do ambiente de testes**

Fonte: O Autor

5.1.1 A configuração dos aplicativos

Depois de concluída a etapa de montagem dos nodos, foram feitas as configurações no sistema operacional Linux e nos aplicativos Heartbeat e DRBD

a) Heartbeat: o primeiro passo foi obter a última versão dos pacotes, no portal do aplicativo em `http://www.linux-ha.org`.

`heartbeat-2.0.7-1.i586.rpm` - pacote base para o funcionamento do aplicativo

`heartbeat-stonith-2.0.7-1.i586.rpm` - mecanismo que provê acesso único ao dispositivo de armazenamento

`heartbeat-pils-2.0.7-1.i586.rpm` - plugins para o Heartbeat

Depois de instalados, foi feita a configuração através do arquivo `/etc/hd.d/ha.cf`.

```
[marcelo@canoeh ha.d]$ cat ha.cf
#       File to write debug messages to
debugfile /var/log/ha-debug

#       File to write other messages to
logfile /var/log/ha-log

#       Facility to use for syslog()/logger
logfacility     local0

keepalive 2

deadtime 120

warntime 150

bcast   eth0            # Linux

auto_failback off

#watchdog /dev/watchdog

#       node    nodename ...    -- must match uname -n
node    nodo01.itflex.local
node    nodo02.itflex.local
```

Figura 18: **Arquivo de configuração `/etc/hd.d/ha.cf`**

Fonte: O Autor

O *keepalive* é o intervalo que os nodos aguardam entre o envio dos *heartbeats*. O *deadtime* indica quanto tempo os nodos backup devem aguardar quando perderem a comunicação com o nodo principal, antes iniciar o processo de *takeover*. A variável *initdead* especifica quanto o nodo deve aguardar para iniciar os serviços, quando for reinicializado por exemplo. Os valores são expressos em segundos e podem variar de acordo com o ambiente. A variável *bcast* in-

forma que o Heartbeat deve usar a interface *eth0* para as mensagens de controle. Nas últimas duas linhas, as váriáveis *node* recebem o nome dos nodos que compõe o cluster.

b) DRBD: foi utilizada a versão 0.8, que teve de ser compilada, para gerar o módulo de acordo com a versão do *kernel* em uso. Este módulo é resposável pela comunicação entre o dispositivo DRBD e os aplicativos.

O particionamento dos discos em ambos os nodos foi feito da conforme a Figura 19.
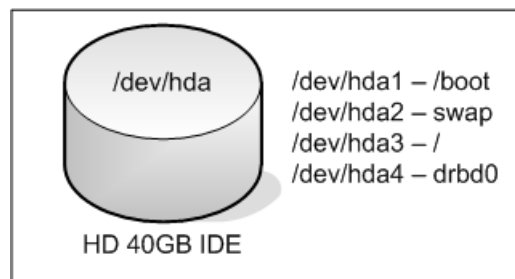
Figura 19: **Partição do disco nos nodos**

Fonte: O Autor

A Figura 20 mostra a configuração do DRBD. Optou-se pelo protocolo C, por ser mais seguro. Foi definido, também, que a partição `/dev/hda4` faria parte do dispositivo DRBD, neste caso o `/dev/drbd0`, os IPs dos nodo e a taxa de transferência máxima que poderia ser usada durante o sincronismo das partições.

A partição `/dev/drbd0` foi formatada usando o sistema de arquivos Terceiro Sistema de Arquivos - *Third Extended File System* (EXT3) e, o ponto de montagem foi definido em `/rede`. O controle também fica por conta do Heartbeat, que monta a partição no nodo que está ativo no momento.

c) Samba: na configuração do samba, foi criado um compartilhamento chamado `publico`, apontando para a partição do DRBD montada em `/rede`. Na seção global da configuração do aplicativo, foi alterada a variável `netbios name = srvcluster` em ambos os nodos. Assim, independente de qual nodo estiver no ar (nodo01 ou nodo02), o cluster sempre será visto como srvcluster.

5.2 A realização dos testes

Após a configuração do ambiente ter sido concluída, foram feitas as primeiras verificações para comprovar que os aplicativos Heartbeat e DRBD estavam funcionando de acordo com o esperado.

```
resource "r0" {
  protocol C;
    startup {
    wfc-timeout       180; ## 3 minutos
    degr-wfc-timeout  180; ## 3 minutos
 }
  disk {
    on-io-error detach;
  }
  syncer {
    rate   10M; # Taxa de transferência máxima
}

  on nodo01.itflex.local {
    device      /dev/drbd0;
    disk        /dev/hda4;
    address     192.168.200.201:7788;
    meta-disk   internal;
}

  on nodo02.itflex.local {
    device     /dev/drbd0;
    disk       /dev/hda4;
    address    192.168.200.202:7788;
    meta-disk  internal;
  }
}
```

Figura 20: **Configuração do DRBD em `/etc/drbd.conf`**

Fonte: O Autor

O primeiro problema encontrado ocorreu devido ao disco do nodo02 ter sido clonado apartir do nodo01. A Fgiura 21 mostra o erro tentar iniciar o Heartbeat. O serviço não era carregado e o arquivo de registro do sistema, o `/var/log/messages` apresentava um erro.

```
nodo01 heartbeat: ERROR: should_drop_message: attempted replay attack [nodo02.itflex.local]?
nodo01 heartbeat: WARN: nodename nodo02.itflex.local uuid changed to nodo01.itflex.local
nodo01 heartbeat: WARN: nodename nodo01.itflex.local uuid changed to nodo02.itflex.local

nodo02 heartbeat: ERROR: should_drop_message: attempted replay attack [nodo02.itflex.local]?
nodo02 heartbeat: WARN: nodename nodo02.itflex.local uuid changed to nodo01.itflex.local
nodo02 heartbeat: WARN: nodename nodo01.itflex.local uuid changed to nodo02.itflex.local
```

Figura 21: **Mensagem de erro ao iniciar o Heartbeat**

Fonte: O Autor

Foi constatado que durante a instalação, o Heartbeat gera um arquivo que o identifica dentro do cluster, ou seja, em cada nodo esse arquivo tem de ser diferente. Pelo fato serem cópias, em ambos os nodos o arquivo era o mesmo, o Heartbeat apontava a inconsistência e avisava que isso poderia até ser uma tentativa de ataque. Para resolver o problema foi necessário apagar o arquivo `/var/lib/heartbeat/hb_uuid`. Ao iniciar o serviço, o Heartbeat criou um novo arquivo e o passou a operar normalmente.

Foi utilizado o aplicativo `tcpdump` para monitorar os pacotes que passam pela interface de rede. Através do `tcpdump` foi possível visualizar o comportamento do Heartbeat em algumas situações. A Figura 22 mostra os pacotes dos *heartbeats*, ou seja, a comunicação feita entre os nodos, que verifica se estão ativos ou não. O nodo01 e o nodo02 através de *broadcast*

enviam pacotes de cerca de 180 bytes que saem da porta 32770 com destino à porta 694 UDP (ha-cluster).

```
IP 192.168.200.201.32770 > 192.168.200.255.ha-cluster: UDP, length 186
IP 192.168.200.202.32770 > 192.168.200.255.ha-cluster: UDP, length 194
IP 192.168.200.202.32770 > 192.168.200.255.ha-cluster: UDP, length 185
IP 192.168.200.201.32770 > 192.168.200.255.ha-cluster: UDP, length 191
IP 192.168.200.201.32770 > 192.168.200.255.ha-cluster: UDP, length 186
IP 192.168.200.202.32770 > 192.168.200.255.ha-cluster: UDP, length 185
```

Figura 22: **Pacotes dos *heartbeats***

Fonte: O Autor

Em seguida foram gravados alguns arquivos no compartilhamento PUBLICO do servidor samba. Pode ser observado os pacotes replicando as informações no nodo02.

```
IP 192.168.200.201.45372 > 192.168.200.202.drbd: P 297:305(8) ack 1568 win 2252 <nop,nop,timestamp 363910 363546>
IP 192.168.200.202.drbd > 192.168.200.201.45372: . ack 305 win 1460 <nop,nop,timestamp 363557 363910>
IP 192.168.200.201.45372 > 192.168.200.202.drbd: P 305:313(8) ack 1568 win 2252 <nop,nop,timestamp 366410 363557>
IP 192.168.200.202.drbd > 192.168.200.201.45372: . ack 313 win 1460 <nop,nop,timestamp 366045 366410>
IP 192.168.200.202.drbd > 192.168.200.201.45372: P 1568:1576(8) ack 313 win 1460 <nop,nop,timestamp 366045 366410>
IP 192.168.200.201.45372 > 192.168.200.202.drbd: . ack 1576 win 2252 <nop,nop,timestamp 366420 366045>
```

Figura 23: **Pacotes de replicação das informações**

Fonte: O Autor

Neste ambiente de testes foi utilizado o protocolo C, que envia pacotes de confirmação depois que os dados foram escritos no nodo secundário, conforme pode ser observado na Figura 24.

```
IP 192.168.200.202.46769 > 192.168.200.201.drbd: . ack 809768 win 16652 <nop,nop,timestamp 34944957 1048976457>
IP 192.168.200.202.46769 > 192.168.200.201.drbd: . ack 812440 win 16652 <nop,nop,timestamp 34944957 1048976457>
IP 192.168.200.202.46769 > 192.168.200.201.drbd: . ack 815336 win 16652 <nop,nop,timestamp 34944957 1048976457>
IP 192.168.200.202.46769 > 192.168.200.201.drbd: . ack 818008 win 16652 <nop,nop,timestamp 34944957 1048976457>
IP 192.168.200.202.46769 > 192.168.200.201.drbd: . ack 822128 win 16652 <nop,nop,timestamp 34944958 1048976457>
```

Figura 24: **Pacotes de confirmação**

Fonte: O Autor

5.3 Teste 1 - forçando o nodo01 a entrar no modo de espera

O pacote do Heartbeat traz alguns scripts, que permitem algumas funcionalidades extras ao aplicativo. Um deles é o `hb_standby`, localizado em `/usr/lib/heatbeat/hb_standby`. Este script força os recursos a migrarem para o servidor secundário. Foi o primeiro teste, para verificar se o recurso de *failover* estava funcionando corretamente.

Conforme pode ser observado na Figura 25, o nodo02 reconhece que o nodo01 quer deixar de ser principal e assumir o a função de secundário. Então todos os serviços disponi-

biliados no nodo01 são parados, é executado o processo de *iptakeover* e *garp* (atualização da tabela ARP dos clientes). Em seguida os serviços são disponibilizados no nodo02.

```
heartbeat: info: nodo01.itflex.local wants to go standby [all]
heartbeat: info: standby: acquire [all] resources from nodo01.itflex.local
eartbeat: info: acquire all HA resources (standby).
ResourceManager info: Acquiring resource group: nodo01.itflex.local IPaddr2::192.168.200.203/24/eth0:0
IPaddr2: INFO: IPaddr2 Resource is stopped
ResourceManager: info: Running /etc/ha.d/resource.d/IPaddr2 192.168.200.203/24/eth0:0 start
IPaddr2: INFO: /sbin/ip -f inet addr add 192.168.200.203/24 brd 192.168.200.255 dev eth0 label eth0:0
IPaddr2: INFO: /sbin/ip link set eth0 up
IPaddr2: INFO: /usr/lib/heartbeat/send_arp -i 200 -r 5 -p.
/var/run/heartbeat/rsctmp/send_arp/send_arp-192.168.200.203 eth0 192.168.200.203 auto 192.168.200.203 ffffffffffff
IPaddr2: INFO: IPaddr2 Success
ResourceManager: info: Acquiring resource group: nodo01.itflex.local drbddisk::r0 Filesystem::/dev/drbd0::/rede::ext3
ResourceManager: info: Running /etc/ha.d/resource.d/drbddisk r0 start
Filesystem: INFO: Running status for /dev/drbd0 on /rede
Filesystem: INFO: /rede is unmounted (stopped)
Filesystem: INFO: Filesystem Resource is stopped
ResourceManager: info: Running /etc/ha.d/resource.d/Filesystem /dev/drbd0 /rede ext3 start
Filesystem: INFO: Running start for /dev/drbd0 on /rede
kernel: EXT3-fs warning: maximal mount count reached, running e2fsck is recommended
Filesystem: INFO: Filesystem Success
ResourceManager: info: Acquiring resource group: nodo01.itflex.local smb
ResourceManager: info: Running /etc/init.d/smb  start
smb: início de smbd succeeded
smb: início de nmbd succeeded
heartbeat: info: all HA resource acquisition completed (standby).
heartbeat: info: Standby resource acquisition done [all].
heartbeat: info: remote resource transition completed.
```

Figura 25: **Forçando os recursos a migrarem para outro nodo**

Fonte: O Autor

5.4 Teste 2 - nodos primário e secundário no ar simultâneamente

Neste teste foi desconectado o cabo de rede do nodo01. Após os 2 minutos sem comunicação, o Heartbeat do nodo01 considerou que o nodo02 estava fora do ar e assumiu os serviços, e vice-versa.

O resultado foi a ocorrência de um problema conhecido como síndrome do cérebro-divido, conforme mostra a Figura . Neste caso, os dois nodos passaram a disponibilizar os mesmos serviços. Ao detectar que os dois nodos estavam no ar, o Heartbeat parou o serviço em ambos, fazendo com que ambos entrassem em um modo de espera.

Em seguida, como os dois nodos estavam indisponíveis, o primário assumiu os serviços. A Figura mostra os registros do sistema, no momento em que isso ocorre.

5.5 Teste 3 - finalizar o *daemon* do samba no nodo primário

Neste teste, foi encerrado o processo do servidor samba no nodo primário, através do comando `killall -9 smbd` e `killall -9 nmbd`, que dessa foram deixou o serviço inacessível. O objetivo era verificar se com o serviço indisponível no nodo01, o Heartbeat iria executar o *failover* e fazer o nodo02 assumir.

```
Mar 31 16:11:49 info: All HA resources relinquished.
Mar 31 16:11:51 info: Received shutdown notice from 'nodo02.itflex.local'.
Mar 31 16:11:51 info: Resource takeover cancelled - shutdown in progress.
Mar 31 16:11:51 info: killing HBREAD process 8426 with signal 15
Mar 31 16:11:51 info: killing HBFIFO process 8424 with signal 15
Mar 31 16:11:51 info: killing HBWRITE process 8425 with signal 15
Mar 31 16:11:51 info: Core process 8425 exited. 3 remaining
Mar 31 16:11:51 info: Core process 8424 exited. 2 remaining
Mar 31 16:11:51 info: Core process 8426 exited. 1 remaining
Mar 31 16:11:51 info: nodo01.itflex.local Heartbeat shutdown complete.
Mar 31 16:11:51 info: Heartbeat restart triggered.
Mar 31 16:11:51 info: Restarting heartbeat.
Mar 31 16:11:51 info: Performing heartbeat restart exec.
Mar 31 16:13:52 WARN: Logging daemon is disabled --enabling logging daemon is recommended
Mar 31 16:13:52 info: **************************
Mar 31 16:13:52 info: Configuration validated. Starting heartbeat 2.0.7
Mar 31 16:13:52 info: heartbeat: version 2.0.7
```

Figura 26: **Dois nodos ativos ao mesmo tempo**

Fonte: O Autor

```
Mar 31 16:13:53 info: Heartbeat generation: 5
Mar 31 16:13:53 info: G_main_add_TriggerHandler: Added signal manual handler
Mar 31 16:13:53 info: Removing /var/run/heartbeat/rsctmp failed, recreating.
Mar 31 16:13:53 info: glib: UDP Broadcast heartbeat started on port 694 (694) interface eth0
Mar 31 16:13:53 info: glib: UDP Broadcast heartbeat closed on port 694 interface eth0 - Status: 1
Mar 31 16:13:53 info: G_main_add_SignalHandler: Added signal handler for signal 17
Mar 31 16:13:53 info: Local status now set to: 'up'
Mar 31 16:13:54 info: Link nodo01.itflex.local:eth0 up.
Mar 31 16:13:56 info: Link nodo02.itflex.local:eth0 up.
Mar 31 16:13:56 info: Status update for node nodo02.itflex.local: status up
Mar 31 16:13:56 info: Running /etc/ha.d/rc.d/status status
Mar 31 16:13:57 info: Comm_now_up(): updating status to active
Mar 31 16:13:57 info: Local status now set to: 'active'
Mar 31 16:13:57 WARN: G_CH_dispatch_int: Dispatch function for read child took too long to execute: 220 ms (> 50 ms) (GSource:
0x80f9c58)
Mar 31 16:13:58 info: Status update for node nodo02.itflex.local: status active
Mar 31 16:13:58 info: Running /etc/ha.d/rc.d/status status
Mar 31 16:14:09 info: remote resource transition completed.
Mar 31 16:14:09 info: Initial resource acquisition complete (T_RESOURCES(us))
Mar 31 16:14:09 INFO: IPaddr2 Resource is stopped
Mar 31 16:14:09 info: Local Resource acquisition completed.
Mar 31 16:14:09 info: Running /etc/ha.d/rc.d/ip-request-resp ip-request-resp
Mar 31 16:14:09 info: Received ip-request-resp IPaddr2::192.168.200.203/24/eth0:0 OK yes
Mar 31 16:14:09 info: Acquiring resource group: nodo01.itflex.local IPaddr2::192.168.200.203/24/eth0:0
Mar 31 16:14:10 INFO: IPaddr2 Resource is stopped
Mar 31 16:14:10 info: Running /etc/ha.d/resource.d/IPaddr2 192.168.200.203/24/eth0:0 start
Mar 31 16:14:10 INFO: /sbin/ip -f inet addr add 192.168.200.203/24 brd 192.168.200.255 dev eth0 label eth0:0
Mar 31 16:14:10 INFO: /sbin/ip link set eth0 up
Mar 31 16:14:10 INFO: /usr/lib/heartbeat/send_arp -i 200 -r 5 -p /var/run/heartbeat/rsctmp/send_arp/send_arp-192.168.200.203 eth0
192.168.200.203 auto 192.168.200.203 ffffffffffff
Mar 31 16:14:10 INFO: IPaddr2 Success
Mar 31 16:14:10 info: Running /etc/ha.d/rc.d/ip-request-resp ip-request-resp
Mar 31 16:14:10 received ip-request-resp drbddisk::r0 OK yes
Mar 31 16:14:10 info: Acquiring resource group: nodo01.itflex.local drbddisk::r0 Filesystem::/dev/drbd0::/rede::ext3
Mar 31 16:14:11 info: Running /etc/ha.d/resource.d/drbddisk r0 start
Mar 31 16:14:11 INFO: Running status for /dev/drbd0 on /rede
Mar 31 16:14:11 INFO: /rede is unmounted (stopped)
Mar 31 16:14:11 INFO: Filesystem Resource is stopped
Mar 31 16:14:11 info: Running /etc/ha.d/resource.d/Filesystem /dev/drbd0 /rede ext3 start
Mar 31 16:14:11 INFO: Running start for /dev/drbd0 on /rede
Mar 31 16:14:12 INFO: Filesystem Success
Mar 31 16:14:12 info: Running /etc/ha.d/rc.d/ip-request-resp ip-request-resp
Mar 31 16:14:12 received ip-request-resp smb OK yes
Mar 31 16:14:12 info: Acquiring resource group: nodo01.itflex.local smb
Mar 31 16:14:12 info: Running /etc/init.d/smb  start
Mar 31 16:14:12 smb: início de smbd succeeded
Mar 31 16:14:12 smb: início de nmbd succeeded
```

Figura 27: **Momento em que o nodo01 assume os serviços**

Fonte: O Autor

O que ocorreu foi que o nodo02 não assumiu. Este comportamento ocorreu porque, mesmo com o serviço estando parado, a comunicação entre os dois nodos continuava funcionando, ou seja, os nodo01 continuava recebendo os *heartbeats* do nodo02 e vice-versa, fazendo o Heartbeat entender que estava tudo nomal com ambos os nodos. O fato de o samba parar foi uma falha de serviço, a qual o Heartbeat não está apto a tratar. Neste caso, se estivesse sendo utilizado um *watchdog*, ele poderia carregar novamente o serviço do samba e tudo voltaria a funcionar normalmente.

5.6 Teste 4 - nodo02 assumindo

O nodo01 estava no ar e o nodo02 estava desligado. O nodo02 foi ligado e logo em seguida o nodo01 falhou, fazendo com que o nodo02 assumisse os serviços.

Como o nodo01 falhou logo após o nodo02 entrar no ar, o tempo não foi suficiente para que o sincronismo dos dados pudesse ser completado. As partições ficaram inconsistentes. Na situação B, o nodo02 mostra os arquivos corretramente. Porém quando o nodo01 assumiu novamente, mostrou os dados que estavam consistentes antes dele falhar. Este comportamento é conhecido como recuperação por retorno, citado na seção 3.5.3 do capítulo 3.
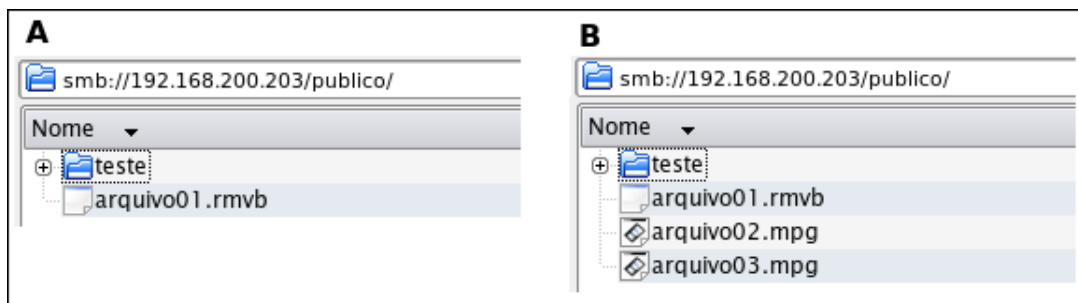
Figura 28: **Inconsistência nos dados**

Fonte: O Autor

Neste caso, o Heartbeat não sincroniza automaticamente as partições, sendo necessária uma intervenção para sincronização manual das partições:

```
[root@nodo02 ~]# cat /proc/drbd
version: 8.0.1 (api:86/proto:86)
SVN Revision: 2784 build by root@nodo02.itflex.local, 2007-03-31 14:18:18
 0: cs:Connected st:Primary/Secondary ld:Consistent
    ns:92968 nr:0 dw:72488 dr:169192401 al:0 bm:4 lo:0 pe:0 ua:0 ap:0
```

5.7 Teste 5 - finalizar processo do DRBD no nodo ativo

Este teste consiste em finalizar o processo do DRBD no nodo ativo através do comando `killall -9 drbd0_worker` ou através do comando `service drbd stop` e tentar descarregar o módulo do *kernel* com o aplicativo sendo usado.

```
[root@nodo02 log]# killall -9 drbd0_worker
[root@nodo02 log]# killall -9 drbd0_worker
[root@nodo02 log]# killall -9 drbd0_worker
[root@nodo02 log]# killall -9 drbd0_worker
[root@nodo02 log]# killall -9 drbd0_worker
[root@nodo02 log]# killall -9 drbd0_worker
[root@nodo02 log]# killall -9 drbd0_worker
[root@nodo02 log]# killall -9 drbd0_worker
[root@nodo02 log]# killall -9 drbd0_worker
[root@nodo02 log]# killall -9 drbd0_worker
[root@nodo02 log]# killall -9 drbd0_worker
[root@nodo02 log]# killall -9 drbd0 worker
[root@nodo02 log]# ps aux | grep drbd
root      4111  0.0  0.0     0    0 ?        S    13:01   0:01 [drbd0_worker]
root      4125  0.0  0.0     0    0 ?        S    13:01   0:07 [drbd0_receiver]
root      4713  0.0  0.0     0    0 ?        S    13:01   0:01 [drbd0_asender]
root     16049  0.0  0.0  3948  604 pts/0    R+   15:46   0:00 grep drbd
[root@nodo02 log]# service drbd stop
Stopping all DRBD resourcesState change failed: (-12) Device is held open by someone
Command '/sbin/drbdsetup /dev/drbd0 down' terminated with exit code 11
drbdsetup exited with code 11
ERROR: Module drbd is in use
.
[root@nodo02 log]# modprobe -r drbd
FATAL: Module drbd is in use.
[root@nodo02 log]# █
```

Figura 29: **Tentando matar o processo do DRBD**

Fonte: O Autor

O processo não deixa ser finalizado através de nenhuma das formas. Essa é uma proteção da aplicação para evitar a inconsistência dos dados.

5.8 Teste 6 - outras verificações

Outros testes executados, que tiveram o mesmo resultado foram:

a) forçar a finalização do processo do Heartbeat através do comando `killall -9 heartbeat`

b) desconectar o cabo de rede do servidor;

c) retirar o cabo de energia do servidor.

Em todas estas situações, não houve mais nenhum tráfego gerado pelo Heartbeat em execução no outro nodo. Assim o nodo que permaneceu ligado entendeu que alguma coisa havia acontecido com a outra máquina e passou a disponibilizar os serviços.

5.9 CONCLUSÃO

Os testes proporcionaram uma melhor compreensão sobre o funcionamento dos aplicativos e no geral os resultados foram satisfatórios.

Tabela 7: **Resultados dos testes**

Teste	**Resultado satisfatório**
Forçando o nodo01 a entrar no modo de espera	SIM
Nodo primário e secundário no ar simultâneamente	SIM
Finalizar o *daemon* do samba no nodo primário	NÃO
Nodo02 assumindo	SIM
Finalizar processo do DRBD no nodo ativo	SIM
Outras verificações	SIM

Fonte: (O Autor)

Porém em duas situações os aplicativos poderiam ter tratado de maneira diferente:

I. sincronismo das partições: no momento em que ambos os nodos entraram no ar novamente depois que a situação ficou estabilizada, o DRBD poderia ter sincronizado as partições automaticamente, pois o nodo01 não mostrou os arquivos que eram esperados. De acordo com o que foi apresentado na tabela 6 do capítulo 3, a recuperação se deu por meio de um retorno ao último estado de consistência (*backward error recovery*), ou seja, este foi o comportamento do aplicativo. Com isso foi necessário forçar o sincronismo manualmente para que as mesmas informações ficassem sincronizadas novamente.

   Levando em consideração que o DRBD pode ser utilizado nos mais diversos ambientes, em que talvez o sicronismo automático não seja requerido, o aplicativo poderia conter uma opção no seu arquivo de configuração, em que o administrador pudesse escolher qual o comportamento esperado numa situação dessas.

II. queda de serviços: como se trata de alta disponibilidade, seria interessante que o Heartbeat tivesse habilidade de monitorar os recuros locais, pois a falha em um serviço não significa que todo o servidor esteja comprometido. Atualmente é necessário o uso de um aplicativo em paralelo, que monitore os recursos e execute ações pré-determinadas caso ocorram problemas.

6 ESTUDO DE CASO

Este capítulo apresenta o estudo de caso de uma transportadora, onde foi instalado um *cluster* de alta disponibilidade utilizando as ferramentas abordadas no capítulo 5. Por questões de segurança, o nome real da empresa não será divulgado. Será utilizado o nome fictício, Levatudo Logística Ltda. Será apresentada a situação da infraestrutura na época, a proposta de melhoria e como ficou o parque após as mudanças.

6.1 O AMBIENTE DA EMPRESA NA ÉPOCA E SUAS NECESSIDADES

A Levatudo Logística Ltda é uma empresa que atua no mercado há 35 anos, oferecendo transportes de carga fracionada, lotações, transportes municipais, intermunicipais, estaduais e interestaduais.

Atualmente a Levatudo Logística Ltda possui uma frota de 109 veículos entre cavalos, carretas e caminhões, distribuídos entre as dez filiais da empresa que estão localizadas nos estados do Paraná, Santa Catarina e São Paulo. Nos demais estados a empresa somente entrega mercadorias através de redespachos feitos por transportadoras parceiras.

A área de TI da empresa apresentava diversos problemas. Os principais estavam relacionados com a disponibilidade e centralização das informações e também conectividade entre matriz e filiais.

A empresa utiliza o aplicativo TCtran para faturamento, emissão de conhecimentos de frete, coletas, entregas, entre outros. Havia uma base de dados principal na matriz e uma base de dados local para cada filial. O sincronismo das informações era feito somente no sentido filiais-matriz, através de arquivos de troca gerados pelo TCtran, que ao final do expediente eram enviados por email para a matriz. O processo de importação dos arquivos era feito manualmente, estava sujeito a erros e frequentemente eram encontradas inconsistências no sistema. Quando as filiais precisavam de algum relatório mais específico, era feita uma solicitação à matriz, que gerava o relatório e disponibilizava via email.

Os servidores de banco de dados eram estações de trabalho que foram montadas usando *hardware* simples, não indicadas para esta tarefa. Era comum estes servidores apresentarem problemas, como travamento ou falha em algum componente de *hardware*. Isso gerava um transtorno muito grande, que atrasava a saída de caminhões. Algumas vezes, até que o sistema entrasse no ar novamente, o processo de liberação da carga era feito manualmente e

depois lançado no sistema, o que acabava por atrasar todo o processo. A Levatudo Logística Ltda estava sujeita a prejuízos financeiros, pois tem contratos com grandes empresas da região e a carga tem de ser entregue dentro do prazo.

Quanto a infraestrutura de comunicação, a Levatudo Logística Ltda utilizava um enlace de Internet ADSL com IP fixo, de 256Kbps. O *firewall* fazia algumas proteções sobre o tráfego que vinha no sentido da Internet em direção a empresa. Não existia nenhum controle sobre o que saia da rede, pois as portas eram todas liberadas. No proxy os controles não eram atualizados, pois não existia nenhum relatório através do qual se pudesse fazer um acompanhamento sobre o que os usuários estavam acessando. Os *emails* da empresa estavam hospedados em um provedor, ou seja, internamente não se tinha controle sobre o que entrava e saia de emails, então se algum funcionário estivesse enviando informações da empresa para fora, não era possível saber quem estava mandando, nem que tipo de informação estava saindo. Além disso, alguns clientes reclamavam que não recebiam os emails de acompanhamento de embarque de mercadoria.

O parque da matriz conta com cerca de 40 estações de trabalho e não existia nenhum controle de banda Qualidade de Serviço - *Quality of service* (QoS) no roteador nem no *firewall*. Com todo esse tráfego, o enlace de Internet também não era mais suficiente. A Figura 30 ilustra melhor o cenário da empresa na época:
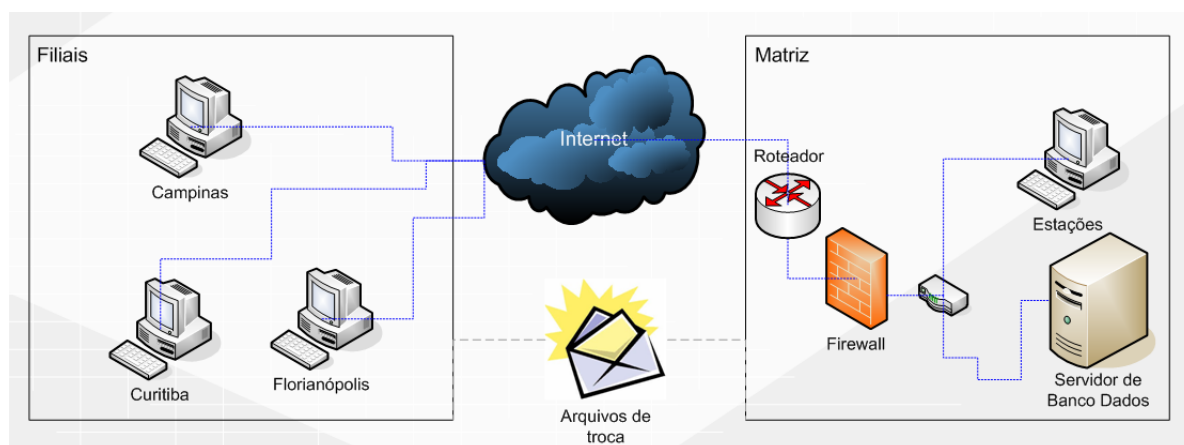
Figura 30: **O ambiente da empresa antes**

Fonte: O Autor

Além dos problemas apresentados, a Levatudo Logística Ltda precisava disponibilizar alguns serviços aos seus clientes, porém com essa infraestrutura não era possível:

I. espaço do cliente: a transportadora precisava disponibilizar no seu portal de Internet, um espaço dedicado aos clientes, onde pudessem verificar faturas, solicitar coletas, consultar

prazos de entrega, emitir segunda via de boleto, etc. O portal proporcionaria conforto tanto para o cliente quanto para a transportadora. O cliente não precisaria mais entrar em contato com a empresa solicitando os serviços. Os colaboradores por outro lado poderiam ser alocados em outras tarefas.

II. acompanhamento de entregas: este é um módulo dentro do espaço do cliente. Os caminhões saiam na parte da manhã fazer entregas. No TCtran a mercadoria somente era dada como entregue a noite, quando eles retornavam para a empresa ou no dia seguinte. Era muito comum clientes ligarem para a transportadora questionando sobre a localização da mercadoria. Não existia nenhum meio de saber a localização da carga, que muitas vezes já havia sido entregue.

Pode-se observar que neste ambiente várias melhorias eram possíveis e necessárias, como por exemplo, implantação de controles sobre todo o tráfego de Internet, contratação de enlaces de Internet com largura de banda maior e disponibilidade das informações para que as filiais tivessem acesso.

6.2 PROPOSTA DE MELHORIA

O projeto de melhoria foi elaborado em conjunto por duas empresas:

a) BC2C Tecnologia: especialista em ambientes Microsoft, é a empresa que já prestava suporte na parte de redes e *hardware* para a Levatudo Logística Ltda.

b) iTFLEX Tecnologia: especialista em soluções de *software* livre, redes e segurança da informação. Foi a empresa contratada para a reestruturação da parte de comunicação.

De início, a Levatudo Logística Ltda disponibilizou dois novos servidores que já haviam sido adquiridos:

```
- 2 servidores HP Proliant 310L
- Processador Intel Xeon 3.2GHz HT (2MB cache)
- 4GB memória RAM
- 2 HDs SCSi 73.4GB 15k RPM - RAID5
```

Por serem equipamentos de alto desempenho, esses servidores foram alocados para tarefas que exigiam mais recursos de hardware.

Foi solicitado solicitado que a transportadora disponibilizasse mais dois servidores, com configuração mais simples, para serem usados como servidor *firewall* e servidor de email.

```
- 1 servidor Intel
- Processador Intel Celeron 2.4GHz
- 512MB memória RAM
- 2 HDs IDE 80GB - RAID1

- 1 servidor Intel
- Processador Pentium IV 3.0 HT
- 512MB memória RAM
- 2 HDs IDE 80GB - RAID1
```

Foi elaborada uma proposta, baseada em quatro etapas principais: reestruturação do CPD da matriz, contratação de novos enlaces de Internet, instalação do *cluster* de firewall e melhorias na infraestrutura de comunicação das filiais.

6.3 A EXECUÇÃO DO PROJETO

Com a proposta aprovada pelo cliente, iniciou-se a primeira etapa do projeto que tratou da reestruturação dos servidores no CPD da matriz:

a) firewall Linux: esse foi um dos pontos mais importantes do projeto. A empresa precisava centralizar seus sistemas de gestão, de modo que as filiais também tivessem acesso. Uma das opções mais comum e financeiramente viável foi através da disponibilização em um servidor, através da Internet. No caso de uma transportadora isso é bastante crítico, pois além de estar disponibilizando dados da empresa e de clientes, ela trabalha com transporte de diversos tipos de mercadoria e cargas de alto valor. Um *firewall* com controles efetivos sobre o tráfego é de extrema importância, para que um possível invasor não tenha acesso a essas informações e consiga por exemplo, descobrir informações sobre rotas dos caminhões, tipos de mercadoria transportados, etc. O novo *firewall* possui controles de acesso internos e externos, seguindo o princípio de que somente as portas de comunicação necessárias são abertas, as demais são fechada. O serviço de proxy controla a navegação e permite a geração de relatórios de acesso por usuário ou estação, VPN para comunicação segura entre matriz e filiais através da Internet e controle de banda (QoS) para otimizar o uso dos enlaces de Internet.

b) servidor Web: nesse novo cenário foi criada uma rede separada, conhecida como Zona desmilitarizada - *DeMilitarized Zone* (DMZ), para os servidores que disponibilizam serviços na Web. A DMZ é criada com o intuito de que se um invasor conseguir acesso privilegiado ao servidor, esse acesso seja somente ao servidor e não a rede interna, onde estão as informações da empresa. Um dos papéis do *firewall* é filtrar essa comunicação. Este servidor, além de

correio eletrônico, disponibiliza webmail, recursos de auditoria sobre as mensagens, antivírus, antispam e FTP. No decorrer do projeto foram disponibilizados também o portal de espaço do cliente, além de um novo módulo do TCtran, o STwap, já comentados anteriormente. O STwap é um aplicativo desenvolvido usando as tecnologias de programação Java a tecnologia de comunicação Protocolo de Aplicações sem Fio - *Wireless Application Protocol* (WAP), que permite aos motoristas interação com o TCtran através de um aparelho de telefone celular. O aplicativo permite que os motoristas dêem baixa nas mercadorias entregues, em poucos minutos a informação é atualizada no TCtran e permite que os clientes saibam que sua mercadoria foi entregue, através do portal do cliente.

c) servidor Windows Terminal: instalado na DMZ e dedicado ao TCtran, onde todas as filiais acessam o aplicativo instalado nele através da VPN. Este servidor usa o sistema operacional Microsoft Windows 2003 Server Terminal Server Edition. Em horários de pico, como quando é feito o faturamento por exemplo, chega a ter 60 usuários simultâneos conectados.

d) servidor de banco de dados: instalado na LAN, este servidor disponibiliza o SGBD (sistema gerenciador de banco de dados) Micrsoft SQL Server 2005, que é utilizado pelo TCtran. Além do banco de dados, esta máquina é utilizada também como servidor de arquivos da rede.

A segunda etapa do projeto tratou da contratação dos enlaces de Internet. Como a reestruturação foi feita pensando em redundância, optou-se por contratar dois enlaces de Internet de operadoras distintas. Foi calculado que cada conexão ao terminal server consumiria cerca de 16Kbps de banda, podendo chegar a 70 conexões simultâneas, ou seja, seria necessário cerca de 1Mbps somente para o sistema. Foram levados em conta também o prazo de atendimento por parte da operadora no caso de falhas com o enlace, largura e garantia de banda e também o custo mensal do serviço. Dentre as propostas apresentadas foram escolhidas duas:

a) Operadora Brasil Telecom: enlace dedicado e redundante de 2Mbps com banda garantida de 50% e 8 endereços IPs (6 válidos); O enlace da operadora Brasil Telecom:

```
endereço da rede: 200.100.20.176/29
roteador Cisco 2600: 200.100.20.177
fw1.levatudo.net - 200.100.20.178
fw2.levatudo.net - 200.100.20.179
mail.levatudo.net - 200.100.20.180
wts.levatudo.net - 200.100.20.181
fw.levatudo.net - 200.100.20.183
```

b) Operadora GVT: enlace ADSL empresarial de 1Mbps com 1 endereço IP válido.

O enlace da Brasil Telecom chega em duas fibras óticas, que percorrem caminhos diferentes da operadora até a transportadora. Caso uma fibra falhe, a outra pode ser usada. Na verdade é como se fossem dois enlaces diferentes, mas usando a mesma faixa de IPs. O contrato firmado com a operadora prevê que o prazo de reparo para problemas que possam ocorrer com o enlace é de 4 horas úteis.

O enlace da GVT é mais simples, oferencendo uma garantia menor de banda e um prazo de reparo maior. Foi contratado com o objetivo principal de prover redundância de comunicação em situações em que o enlace principal estiver indisponível.

Ambos os enlace foram ligados no *firewall* e utilizam um recurso do sistema operacional Linux conhecido como *policy routing* (roteamento por política), disponibilizado pelo pacote `IProute2`, que permite escolher quais serviços farão uso de qual enlace. Foram definidas três situações:

I. operação normal: dois enlaces funcionando. Brasil Telecom dedicado ao TCtran e servidor de correio eletrônico. GVT dedicado a navegação.

II. enlace da Brasil Telecom fora do ar: todo o tráfego direcionado para o enlace da GVT, ou seja, todos os serviços passam a ser disponibilizados neste enlace. Além das regras de *policy routing*, os apontamentos de DNS também são alterados e em 3 minutos passam a apontar para o IP da GVT.

III. enlace GVT fora do ar: o tráfego de navegação passa também para o enlace da Brasil Telecom.

Após a ativação de ambos os enlaces, foi feita a configuração do *QoS* no *firewall*. Foi necessário a aplição de *patches* (adição de uma pequena parte de código fonte) ao *kernel* do sistema operacional Linux e ao pacote *iptables*, que agregaram suporte ao *Qos* tanto sobre o tráfego de entrada quanto o de saída.

O enlace foi dividido em duas classes:

a) classe I, com reserva de 75% (1,5Mbps) do enlace para uso com o sistema TCtran;

b) classe II, utilizando os 25% (512Kbps) restantes para tráfego de emails, FTP e demais aplicações.

O sistema operacional Linux permite que este controle seja dinâmico, ou seja, se uma classe não estiver fazendo uso de toda largura de banda reservada a ela, a banda que sobra pode ser emprestada para uma classe vizinha.

A terceira parte do projeto tratou da instalação do firewall de alta disponibilidade (fw.levatudo.net). Neste caso foi utilizado somente o aplicativo Heartbeat, pois as informações nos discos mudam com pouca freqüência, não sendo necessária a utilização do DRBD. O sincronismo do sistema de arquivos é feito semanalmente através do aplicativo `rsync`.

A instalação do nodo *backup* (fw2.levatudo.net) foi feita através de um clone do HD do *firewall* (fw1.levatudo.net) que já estava no ar e foi usado como nodo primário. A clonagem foi feita através do aplicativo `rsync`. Concluída esta tarefa, foram ajustados os parâmetros de configuração, tais como endereçamento IP e nome do servidor.

Com os dois nodos no ar, foi feita a instalação e configuração do Heartbeat. Os recursos sob controle do Heartbeat são: um script que avisa os administradores por email caso ocorra alguma falha e os endereços de IP virtual (firewall e terminal server), que são IPs válidos. Existe um NAT para o IP inválido dos servidores que estão na DMZ.

Cada nodo possui quatro interfaces de rede. Além disso existem 2 *switches*, um para a ligação dos enlaces nas interfaces WAN e outro para ligação dos servidores que estão na DMZ.

A Figura 31 ilustra o ambiente do *cluster* depois de pronto.

Das nove filiais, na época que foi iniciado o projeto somente duas tinham *firewall* e não tinham nenhuma relação com a matriz. Atualmente sete delas já possuem *firewall* semelhante ao da matriz, que além da proteção, disponibilizam os serviços de DHCP, proxy, VPN e *QoS*, permitindo assim que as filiais fechem um túnel de comunicação segura com a matriz através da Internet e proporcionando um acesso rápido ao sistema.

6.4 OS RESULTADOS ALCANÇADOS

Para que fosse realizada toda essa mudança que atingiu a matriz e filiais, a Levatudo Logística Ltda fez um investimento alto. Porém as melhorias e os benefícios puderam ser sentidos logo no início.

I. centralização: os novos servidores de banco de dados e do aplicativo TCtran, permitiram a centralização das informações em um único banco de dados, elimitando completamente as inconsistências que ocorriam anteriormente, com os arquivos de troca e o processo de importação manual. Através da VPN, todas as filiais acessam a mesma base de dados, o que permite um acompanhamento em tempo real das atividades que acontecem na empresa, como por exemplo faturamento, saída de caminhões e entregas.
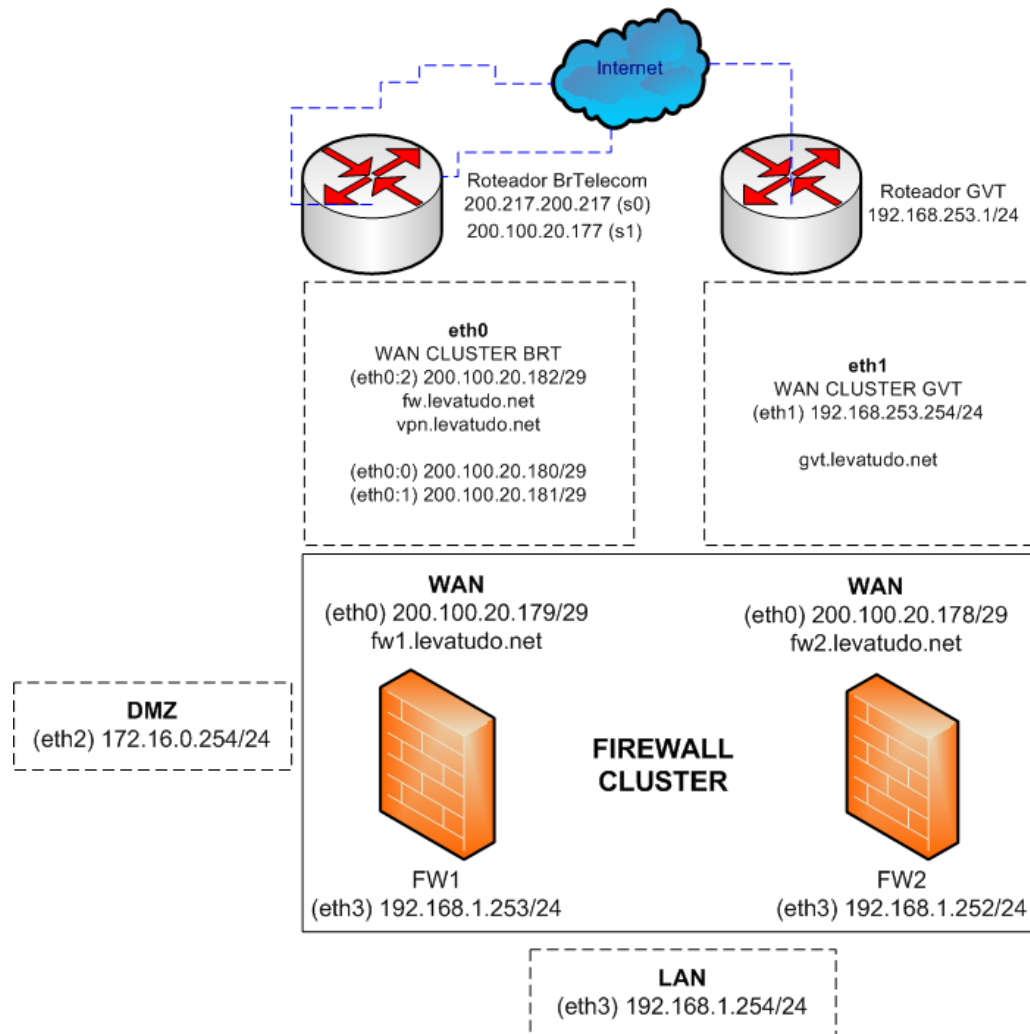
Figura 31: **Ambiente do *cluster***

Fonte: O Autor

II. segurança: a nova estrutura proporciona uma maior confiabilidade nos dados disponibilizados pelo sistema, uma vez que as inforamções são gravadas e consultadas no mesmo local. A VPN, utilizando recursos de criptografia permite a comunicação segura das filiais com a matriz através de uma rede insegura, que é a Internet. Além disso, o novo *firewall* proporciona um forte controle sobre o tráfego que entra e sai da rede da empresa.

III. produtividade: a nova estrutura proporcionou de modo geral uma maior produtividade na transportadora. Uma vez que as informações estão centralizadas, os processos que envolvem a matriz e as filiais correm de maneira mais rápida. O portal do cliente disponibiliza muitos recursos, através dos quais o próprio cliente pode interagir com a empresa, poupando tempo e alocação de colaboradores internos.

IV. disponibilidade: o *firewall cluster* visa garantir a disponibilidade das informações a maior parte do tempo possível, de modo que a qualquer momento possam ser acessíveis pelas

filiais ou pelos clientes. Indisponibilidade pode ser sinônimo de caminhões parados e prejuízos.

A Figura 32 mostra a disponibilidade do *firewall cluster* do dia 01/01/2007 até 27/05/2007. Houve um momento em que ambos os nodos falharam, ficando fora do ar por cerca de três horas. Isso mostra que a disponibilidade até o dia 27 era de 99,8%.

State Breakdowns:

Up : (99.822%) 146d 10h 46m 31s

Down : (0.101%) 0d 3h 32m 59s

Unreachable : (0.078%) 0d 2h 44m 6s

Indeterminate: (0.000%) 0d 0h 0m 0s
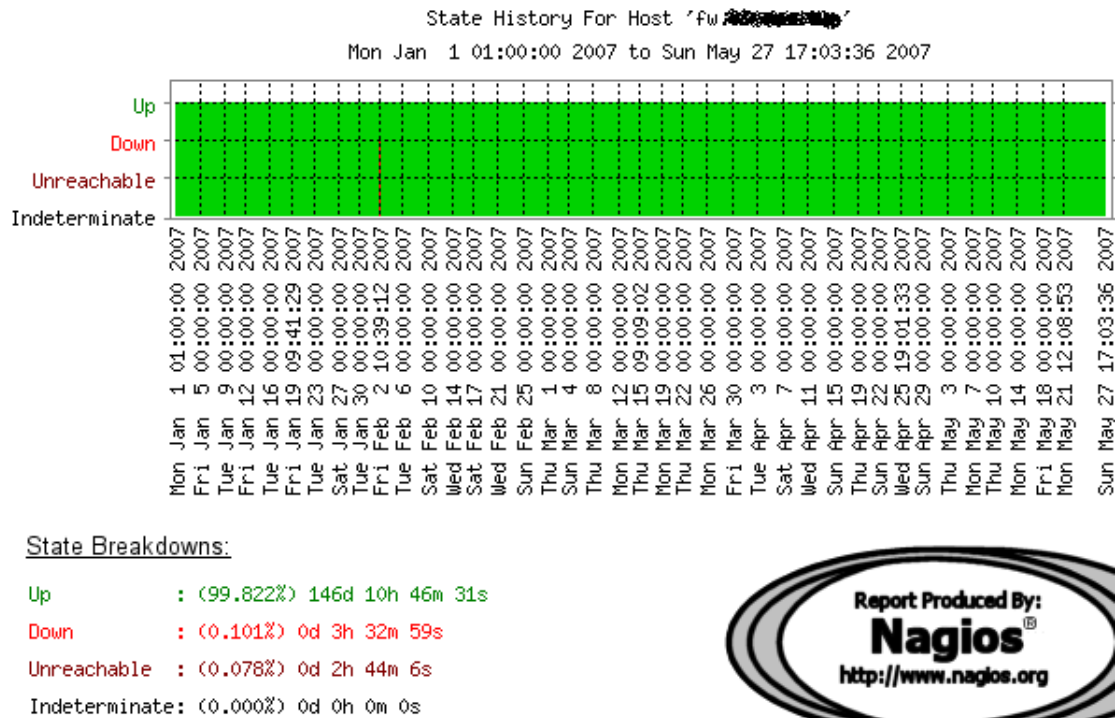
Figura 32: **Disponibilidade do *firewall* no ano de 2007**

Fonte: O Autor

V. escalabilidade: a nova infraestrutura já foi preparada pensando em agregar novos recursos, entre eles o uso de VoIP (voz sobre IP) através da Internet, para ligações telefônicas entre as unidades da empresa com custo zero.

7 CONCLUSÃO

Este trabalhou iniciou apresentando a história do sistema operacional Linux, e definindo os *clusters* e as suas características. Logo após foi abordada a alta disponibilidade e tolerância à falhas, em que foram apresentados os principais termos e classificações usadas. É muito importante que a parte teórica de alta disponibilidade seja bem definida, pois nessa área muitos termos iguais são utilizados para conceitos diferentes. Com os conceitos definidos, foram analisadas e testadas as ferramentas que permitem a criação de um ambiente de alta disponibilidade. A parte teórica dos capítulos 2 e 3 foi fundamental para que se pudesse compreender o funcionamento e, durante os testes, entender o comportamento dos aplicativos apresentados no capítulo 4.

As soluções comerciais de alta disponibilidade geralmente implicam em altos investimentos, pois no momento em que é feita a compra, o cliente paga por um pacote, o que acaba por deixá-lo sem opção de escolha, ou seja, o cliente é obrigado a comprar o *hardware*, os aplicativos e o serviço do mesmo fornecedor, para garantir que seu problema possa ser resolvido.

O sistema operacional Linux por outro lado, permite a criação soluções de alta disponibilidade utilizando *hardware* simples, aplicativos livres de fácil implementação e que podem ser utilizados em conjunto com outras aplicações.

Tanto o Heartbeat quanto o DRBD atenderam as expectativas durante a implementação. Atualmente estas soluções vêm sendo utilizadas em diversas áreas e têm tomado o espaço das soluções comerciais, não somente por ter um baixo custo de implementação, mas por se mostrarem bastante flexíveis na configuração e robustas durante o uso, caracteríticas que são importantes em um ambiente de alta disponibilidade. O único ponto em que o Heartbeat deixou a desejar, foi na questão de monitoramento de recursos locais, mas segundo os desenvolvedores, em breve esta característica estará presente.

O projeto proporcionou uma grande comodidade e satisfação, tanto para a transportadora que conseguiu melhorar a infraestrutira de TI como um todo, quanto para o cliente, que é atingido também.

O mercado está cada vez mais competitivo e os clientes cada vez mais exigentes, por isso é fundamental que a empresa possa oferecer um bom serviço, com entregas rápidas e um baixo custo, afim de garantir sua posição no mercado. Os recursos implantados nesse projeto para a Levatudo Logística Ltda representam um passo nessa direção.

Para o futuro existe a proposta de montar um *cluster* para o servidor web, pois o mesmo disponibiliza aplicações importantes para a empresa, que requerem disponibilidade e, existe também a hipótese se se montar um *cluster* de servidores *Windows*. Atualmente os serviços de banco de dados e o TCtran podem ser disponibilizados no mesmo servidor, caso algum falhe, porém isso requer intervenção de um técnico.

Uma das dificuldades encontradas no desenvolvimento do trabalho foi a busca por referências. O tema *clusters* de alta disponibilidade ainda é pouco explorado se comparado a outras áreas, como por exemplo, os *clusters* de alto desempenho. Grande parte do material encontrado, como artigos por exemplo, são escritos por alguns poucos autores.

Algumas sugestões para trabalhos futuros são:

I. implementação do Heartbeat em um ambiente com mais de dois nodos, afim de se observar como o Heartbeat trata o processo de *failover* nesse ambiente;

II. utilizar o DRBD no modo leitura/escrita em mais de dois nodos simultaneamente e verificar a consistência dos dados;

III. estudo comparativo de desempenho entre sistemas de arquivos distribuídos, tais como DRBD, GFS e OCFS2;

IV. estudo comparativo entre diferentes ferramentas de monitoramento, afim de se identificar qual delas é a melhor para trabalhar em conjunto com o Heartbeat e o DRBD.

GLOSSÁRIO

API Interface para Programação de Aplicações - *Application Programing Interface*

CENAPAD Centros Nacionais de Processamento de Alto Desempenho

DMZ Zona desmilitarizada - *DeMilitarized Zone*

DRBD Dispositivo de bloco de armazenamento distribuído - *Distributed Replicated Block Device*

EXT3 Terceiro Sistema de Arquivos - *Third Extended File System*

FSF Fundação do Software Livre - *Free Software Foundation*

GCC Compilador GNU C - *GNU C Compiler*

GPL Licença Pública Geral - *General Public License*

GFS Sistema de Arquivos Global - *Global File System*

HAC Clusters de alta disponibilidade - *High Availability Clusters*

HPC Clusters de alto desempenho - *High Performance Clusters*

INPE Instituto Nacional de Pesquisas Espaciais

LBC Clusters de balanceamento de carga - *Load Balance Clusters*

LINUX-HA *Linux High-Availability*

LSB Base Padrão do Linux - *Linux Standard Base*

MIT Instituto de Tecnologia de Massachusetts - *Massachusetts Institute of Technology*

NIST Instituto Nacional de Padrões e Tecnologia dos Estados Unidos - *National Institute of Standards and Technology*

OCF Padrão de Cluster Aberto - *Open Cluster Framework*

OCFS2 Sistema de Arquivos Oracle versão 2 - *Oracle Cluster File System version 2*

PETROBRAS Petróleo Brasileiro S.A.

QoS Qualidade de Serviço - *Quality of service*

SPOF Ponto Único de Falha - *Single Point of Failures*

SSH *Secure Shell*

STONITH Atire na Cabeça do Outro nodo - *Shot The Other Node In The Head*

TCP/IP Protocolo de Controle de Transmissão/Protocolo de Internet - *Transmission Control Protocol/Internet Protocol*

TELEBRAS Telecomunicações Brasileiras S.A.

TI Tecnologia da Informação

VPN Rede Virtual Privada - *Virtual Private Network*

WAP Protocolo de Aplicações sem Fio - *Wireless Application Protocol*

REFERÊNCIAS

ABREU, H. O. e. a. **Construindo cluser beowulf com software livre** . Manaus: Monografia apresentada na UNAMA, 2004.

ALMEIDA, L. **Entraves à Dependabilidade**. Aveiro: Monografia apresentada na Universidade de Aveiro, 2001.

AMDAHL, G. M. **Validity of the single processor approach to achieving large scale computing capabilities**. California: Artigo publicado na IBM, 1967.

BRASILEIRO, F. V. **Seljuk: Um ambiente para suporte ao desenvolvimento e à execução de aplicações distribuídas robustas**. Campina Grande: Monografia apresentada na UFPB, 2000.

CENAPAD. **Guia do Usuário CENAPAD**. 2007. Unicamp. Disponível em: <http://www.cenapad.unicamp.br>.
Acesso em: 20 fev. 2007.

DARWIN, I. F. e. a. **Linux Standard Base Core Specification 3.1**. 2006. The Free Standards Group. Disponível em: <http://www.linux-foundation.org/en/Specifications>.
Acesso em: 19 abr. 2007.

DEAN, J. e. a. **Web search for a planet: The google cluster architecture**. Mountain View: Artigo publicado pelo IEEE, 2003.

FAUSTINO, E. P. e. a. **Construindo supercomputadores com Linux**. Goiânia: Monografia apresentada no Cefet, 2006.

FERREIRA, L. e. a. **Linux HPC Cluster Installation**. Austin: IBM, 2001.

FRANCO, L. D. **Implementação computacional em ambiente paralelo de memória distribuída para análise acoplada de sistemas off shore**. Rio de Janeiro: Monografia apresentada na UFRJ, 2004.

GARCIA, S. **Clusters de Balanceamento de Carga em Linux**. São Paulo: Artigo publicado na Slackwarezine, 2007.

KOPPER, K. **The Linux Enterprise Cluster**. San Francisco: No Starch Press, 2005.

LAUREANDO, S. C. **Un cluster in alta disponibilità in ambiente Linux**. Bari: Monografia apresentada a Università Degli Studi di Bari, 2004.

MANFREDINI, C. **Medida previsória do bem**. 2005. Ministério do Desenvolvimento. Disponível em: <http://www.desenvolvimento.gov.br/sitio/ascom/noticia.php?cd_noticia=6454>.
Acesso em: 18 mar. 2007.

MARCO, L. M. d. **Computação de Alto Desempenho: Clusters**. Artigo publicado pela Unicamp, Campinas, 2006.

MISAGHI, M. **Introdução a Gerência de Redes**. Joinville: Notas de aula utilizadas no IST, 2006.

MUSSI, E. e. a. **Guia de Estruturação e Administração do Ambiente de Cluster e Grid**. Brasília: Artigo publicado pelo Ministério do Planejamento, 2007.

NEMETH, E. e. a. **Manual Completo do Linux: Guia do Administrador**. São Paulo: Makron Books, 2004.

OYAMADA, M. S. e. a. **Alta disponibilidade utilizando sistemas Linux**. Cascavel: Artigo publicado pela UNIOESTE, 2003.

PARKER, P. M. **Definition: Cluster**. 2007. Webster's Online Dictionary. Disponível em: <http://www.websters-online-dictionary.org/definition/cluster>. Acesso em: 08 maio. 2007.

PEREIRA, N. A. **Serviços de pertinência para clusters de alta disponibilidade**. São Paulo: Dissertação de mestrado apresentada na USP, 2004.

PITANGA, M. **Construindo Supercomputadores com Linux**. Rio de Jaineiro: Brasport, 2002.

REISNER, P. **DRBD**. Wien: Artigo publicado pela Linbit, 2002.

REISNER, P. **Rapid resynchronization for replicated storage Activity-logging for DRBD**. Wien: Artigo publicado pela Linbit, 2004.

RESNICK, R. I. **A Modern Taxonomy of High Availability**. 1996. General Concepts. Disponível em: <http://www.generalconcepts.com/resources/reliability/resnick/HA.htm>. Acesso em: 28 mar. 2007.

ROBERTSON, A. **The evolution of the Linux-HA project**. Austin: Artigo publicado pela IBM, 2004.

SANTOS, A. C. **Tolerância a falhas para sistemas embarcados**. Recife: Monografia apresentada na UFPE, 2000.

SCHIOZER, D. J. **Computação paralela aplicada à simulação numérica de reservatórios**. Campinas: Monografia apresentada na Unicamp, 1997.

STALLMAN, R. **The Free Software Definition**. 2007. Free Software Fundation. Disponível em: <http://www.fsf.org/licensing/essays/free-sw.html>. Acesso em: 08 abr. 2007.

TESTONI, A. G. S. **Soluções RAID para prevenção de falhas ocorridas em discos rígidos de computador**. Joinville: Monografia apresentada no IST, 2005.

TOSATTI, M. **High Availability**. 2006. Artigo publicado pelo Kernel Newbies. Disponível em: <http://kernelnewbies.org/Documents/HighAvailability>. Acesso em: 08 jan. 2007.

VARGAS, E. **High Availability Fundamentals**. Palo Alto: SUN, 2000.

VOGELS, W. e. a. **The design and architecture of the Microsoft Cluster Service**. Piscataway: Artigo publicado pelo IEEE, 1998.

WATSON, M. e. a. **High Availability on the RISC System/6000 Family**. Austin: IBM, 1995.

WEBER, T. S. **Um roteiro para exploração dos conceitos básicos de tolerância a falhas**. Porto Alegre: Artigo publicado na UFRGS, 2002.

WIKIPEDIA. **Computer cluster**. 2006. Wikipedia. Disponível em: <http://en.wikipedia.org/wiki/ARCnet>.
Acesso em: 10 jan. 2007.